RL-TR-96-204
Final Technical Report
June 1997

PLANNING IN COMPLEX WORLDS VIA MIXED-INITIATIVE INTERACTION

University of Rochester

James F. Allen, Lenhart K. Schubert, and George M. Ferguson

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Rome, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-96-204 has been reviewed and is approved for publication.

APPROVED:

NORTHRUP FOWLER, III
Project Engineer

FOR THE COMMANDER:

JOHN A. GRANIERO, Chief Scientist
Command, Control, & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/C3C, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1997	Final Feb 91 - Jan 96

4. TITLE AND SUBTITLE
PLANNING IN COMPLEX WORLDS VIA MIXED-INITIATIVE INTERACTION

5. FUNDING NUMBERS
C - F30602-91-C-0010
PE-62702F
PR-5581
TA -27
WU-58

6. AUTHOR(S)
James F. Allen, Lenhart K. Schubert, and George M. Ferguson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Computer Science
University of Rochester
Rochester, NY 14627

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3C)
525 Brooks Rd
Rome, NY I344I-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-96-204

11. SUPPLEMENTARY NOTES
Rome Laboratory Engineer: Northrup Fowler III /C3C/(315) 330-3011

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This final report presents an overview of the research performed at the University of Rochester under Rome Lab contract F30602-91-C-0010. The project addressed problems in developing large-scale plans in complex worlds. The results can be divided into four general areas.

First, we addressed some fundamental representational issues by developing new representations of actions and plans that increase the expressiveness of plan representations.

Second, we addressed efficiency issues by developing a set of temporal reasoning algorithms for the efficient handling of very large-scale temporal databases.

Third, we developed a new set of heuristic search methods that greatly improve the efficiency of well-founded planning systems such as UCPOP. Several of these techniques have been incorporated into the latest version of UCPOP distributed by the University of Washington.

Fourth and finally, we addressed the problem of developing plans in the real world by constructing and testing a new model of mixed-initiative planning.

The model of mixed-initiative planning may be the result that has the most lasting impact on the field. By viewing the human as an essential part of the planning process, we dramatically change the problems that are important for the ultimate successful application of planning technology.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Planning, Mixed-Initiative Planning, Plan representation, temporal reasoning, heuristic search	166
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFFIED	UNCLASSIFIED	UL

Contents

List of Figures

1. Executive Summary

This paper presents an overview of the research performed at the University of Rochester under Rome Lab contract F30602-91-C-0010. The project addressed problems in developing large-scale plans in complex worlds. The results can be divided into four general areas.

First, we addressed some fundamental representational issues by developing new representations of actions and plans that increase the expressiveness of plan representations. More specifically:

- We developed an expressive logic of time and action for planning that supports reasoning in domains with external events and interacting simultaneous actions.
- We developed a new approach to the frame problem called explanation closure, and showed its utility in two planning frameworks: one based on the situation calculus and the other, interval temporal logic.
- We developed a new representation for plans based on argument structures that provides a firm foundation for future work in mixed-initiative planning where plans need to be negotiated between two or more agents.

Second, we addressed efficiency issues by developing a set of temporal reasoning algorithms for the efficient handling of very large-scale temporal databases. More specifically:

- We performed an evaluation of five different temporal reasoning systems and studied their behavior as the size of the temporal data increased (up to 60,000 time periods).
- We explored the theoretical foundations of point-based temporal reasoning systems (with and without disjunction), and developed theoretically sound methods of reasoning about large sets of temporal information.
- Based on these theoretical developments, we built a new temporal reasoning system, TimeGraph II, that improved on the best systems in the evaluation by a factor of two or more on large databases.

Third, we developed a new set of heuristic search methods that greatly improve the efficiency of well-founded planning systems such as UCPOP. Several of these techniques

have been incorporated into the latest version of UCPOP distributed by the University of Washington.

Fourth and finally, we addressed the problem of developing plans in the real world by constructing and testing a new model of mixed-initiative planning. More specifically:

- We defined a dialogue-based framework for mixed-initiative planning, which is the first planning model that takes the human planner as an essential part of the planning process.
- We implemented a first prototype system, TRAINS-95, that demonstrated the dialogue-based model in a simple route planning domain and could be used by people with virtually no training.
- We performed the first usability-based evaluation of a planning system, testing how well the system actually helps people plan. While a preliminary first step, we view this type of evaluation playing a vital role in mixed-initiative planning research in the future.

The model of mixed-initiative planning may be the result that has the most lasting impact on the field. By viewing the human as an essential part of the planning process, we dramatically change the problems that are important for the ultimate successful application of planning technology.

2. Introduction

In command and control situations and logistics planning, a human planner faces several difficult problems. First, there is a surplus of data, only a small amount of which is actually relevant to the current task. In fact, what data is relevant cannot be determined in advance and only becomes clear as the situation and the plan develop. Second, the plans being considered are large and complex, and it is beyond human capabilities to manage all the details effectively. Automated planning systems are better able in principle to handle the scale, but are hard to apply because of the under-specified initial situations, and the fact that many planning decisions are made on an intuitive basis that cannot be effectively quantified. As a result, neither the human nor the computer can effectively solve such planning problems in isolation.

This problem motivates the three research areas which have been the focus of the work at Rochester:

1. Mixed-initiative planning systems, where the computer acts as a collaborating assistant to the human. By cooperating, the human-computer "team" is able to deal with problems that neither could handle easily alone.

2. Plan representation formalisms that go beyond the assumptions underlying most planning formalisms and handle such complexities as external events and interacting overlapping actions.

3. Efficient algorithms for handling large-scale problems, especially in dealing with large-scale temporal databases, and in developing heuristics for speeding up traditional "well-founded" planners.

2.1 Mixed-Initiative Planning

The TRAINS Project at the University of Rochester [Allen *et al.*, 1995] is a long-term research effort to develop intelligent assistants that interact with their human managers using natural language. To explore mixed-initiative planning, we designed and built a prototype system, TRAINS-95, that helps a manager solve routing problems in a simple transportation domain. The manager is presented with a map displaying cities and rail connections between

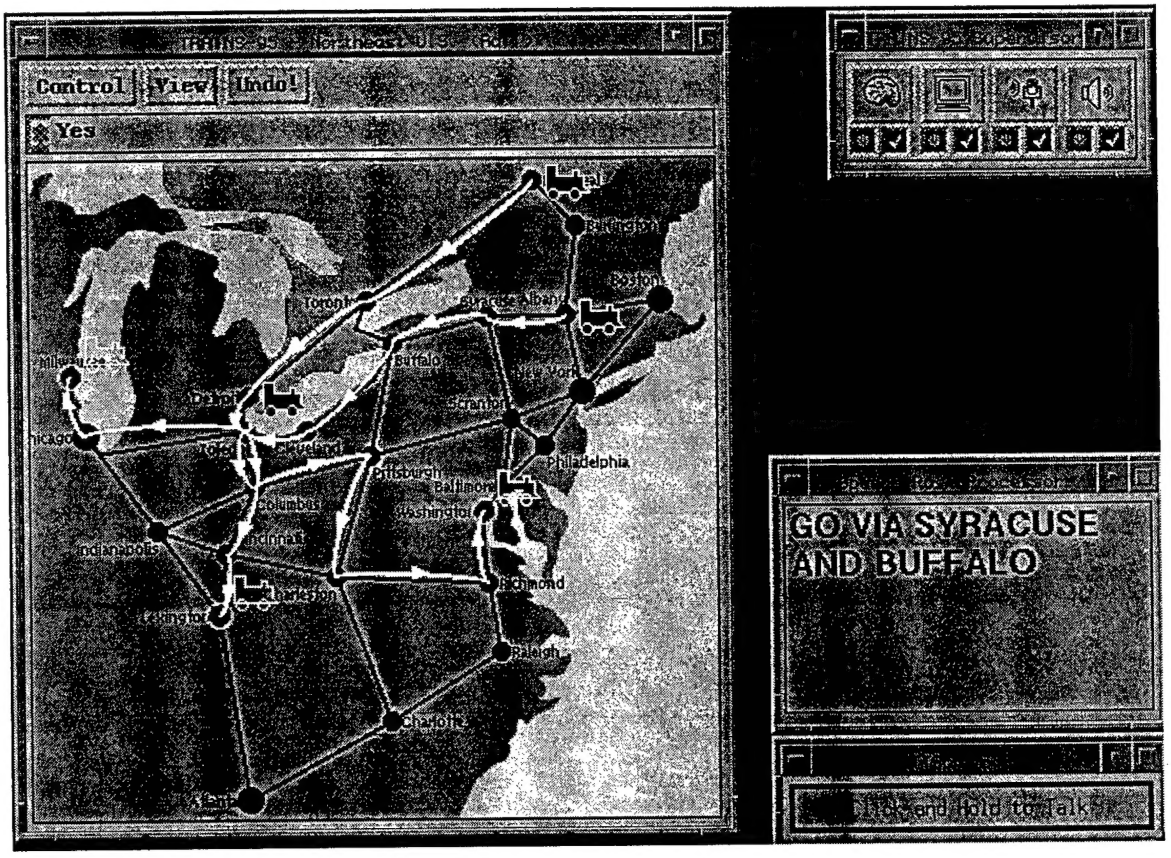


Figure 2.1: Sample screen from the TRAINS-95 System, showing map display, speech recognition panel, and system controller

them. A sample screen display is shown in Figure 2.1. The system generates random problems that require planning routes for a set of engines to a set of destinations. Various environmental factors can arise during the interaction, which the manager and system must then plan to accommodate.

Our goal was a robust, modular, multi-modal, mixed-initiative planning assistant. By "robustness" we mean that no matter what occurs during the interaction, the system not only doesn't crash, but does something to indicate its understanding of the manager's intentions and its own attempts to further the plan. By "modular" we are taking seriously the idea that there are, or will be shortly, a variety of knowledge sources, reasoning agents, and display engines available as resources that the system can employ. Examples include weather information, news feeds, map servers, and so on, as well as "off-the-shelf" technology such as speech recognizers and generators. By "multi-modal" we mean that there are a multitude of ways of communicating between humans and computers, include speech input and output, written text, and graphical displays such as maps, charts, forms, and the like, for both input and output. By treating all modalities as *linguistic*, that is, as a form of language, we obtain a powerful unifying model of the interaction as a form of *dialogue*.

Finally, by "mixed-initiative" we mean that both the system and the human are on roughly equal ground as participants in the dialogue. This is not to say that they are equals, since clearly they are good at different things. But in a truly mixed-initiative system, both participants can do what they do best. The human typically has knowledge of the high-level goals and means of achieving them, while of course, the computer is good at managing the multitude of low-level details that go into such plans. As well, in realistic situations, the human may be guided by principles that are difficult or impossible to quantify precisely, making it impossible to fully automate the task and replace them. Even if this were possible in principle, the resulting master-slave style of communication would hardly be natural and probably wouldn't be very efficient.

2.2 Representing Time, Action and Plans

Most traditional representations of action and plans use a state-based approach that builds in strong assumptions about the nature of action and the world (the so-called STRIPS assumptions). These require that action definitions completely describe how the action changes the world, and allows only a single action to be performed at a time.

Our work focused on using an expressive temporal logic to generalize this model and develop models of actions, events and plans to support reasoning in complex worlds. There were five key points that we felt were essential for such a representation:

1. Actions and events take time. While some events may be instantaneous, most occur over an interval of time during which other events and actions may also occur.

2. The relationship between actions and their effects is complex. Some effects become true at the end of the action, but others become true at the beginning and still others may be true during the action and not true after.

3. Actions and events may interact in complex ways when they overlap, including the production of additional effects (synergy) and partial or full interference.

4. Externally-caused changes may occur no matter what the agent plans. The planner must be able to reason about possible external events so as to construct reliable plans.

5. Knowledge of the world and the possible actions is incomplete in most applications. Virtually no plan is thus foolproof and can only be made on the basis of certain assumptions, which should be made explicit.

We developed a representation [Allen and Ferguson, 1994] that is significantly more expressive and more natural than previous approaches along these criteria.

We also developed a new approach to the frame problem called *explanation closure* [Schubert, 1994] that can be used both with more traditional representations and with the interval temporal logic. Explanation closure makes the assumptions underlying the frame problem explicit, leading to a much richer representation than possible using the STRIPS assumption or non-monotonic models that minimize change.

2.3 Reasoning About Large-Scale Problems

No matter what plan representation is used, it is clear that efficient temporal reasoning will be an essential part of any system dealing with complex planning problems. Early in the project, we performed an evaluation of six existing temporal reasoning systems on constructed databases reflecting movement in the TRAINS world [Yampratoom and Allen, 1993]. These databases ranged from a few hundred temporal elements up to 60,000. We found that the expressive interval-based temporal reasoners could not handle databases of more than a few hundred times effectively, and completely collapsed around 500 temporal elements. The less expressive point-based reasoners fared better, but those that used data structures encoding a completely connected set of constraints eventually became unusable on the databases involving tens of thousand elements. The most promising approaches were the systems that did not construct a complete constraint graph, such as Schubert'sTimeGraph system, which is further discussed below.

These results motivated further development both to improve the performance of reasoners over the "time point algebra" (relations using $<$, $\leq$, $=$, $\neq$) and to develop efficient methods for handling more expressive representations allowing disjunctions.

Another problem in dealing with large-scale problems is the search efficiency of traditional planners. We concentrated on "well- founded" planning methods, such as UCPOP, that are

sound and complete and have other desirable properties and developed a set of methods for improving the performance of such systems by an order of magnitude. The techniques range from modifying the search strategy to include necessary actions first, to preprocessing methods that restricts that set of values that a variable can take.

The remainder of this paper describes these results in more detail. We discuss mixed-initiative planning and the TRAINS-95 system in Section 3, our work on the representation of actions, events, and plans in Section 4, our work on efficient large-scale temporal reasoning in Section 5, and our work on improving the efficiency of well-founded planning algorithms in Section 6. While problems remain, our results make a significant contribution to the goal of building mixed-initiative systems for constructing large-scale, realistically complicated, plans.

3. The TRAINS-95 System

The TRAINS-95 system was built to demonstrate the feasibility and usefulness of dialogue-based models of mixed-initiative planning. The system, which runs in near real-time and supports speech, keyboard and a map display.[1] The key insights motivating this work were:

1. that the dialogue should provide the context required for successful interaction independent of input modality; and

2. that the plan reasoning requirements of mixed-initiative systems differ markedly from the specifications of traditional planners.

The domain reasoner in TRAINS-95 maintains a knowledge base describing the state of the world and provides planning and plan recognition services to the dialogue modules. For the simple route-planning domain, of course, it would be easy to build a perfect reasoner that solved the problems as soon as the manager had stated their goals. However, it is unlikely that we will ever be able to build such a reasoner for a realistic domain. We therefore deliberately weakened the TRAINS-95 domain reasoner so to force the manager to interact in order to overcome its shortcomings. The route planner can therefore only plan route segments less than four hops long, and for those it chooses a random path. The knowledge base maintains an accurate view of the map, and allows various "natural" events such as bad weather or track maintenance to arise during the interaction. These also force interaction in order to revise plans to take account of them.

The domain reasoning in TRAINS-95 is incremental and incorporates aspects of both planning and plan recognition in a tightly-coupled way. For example, the domain reasoner may be asked to incorporate a new constraint on an existing plan, *e.g.*, that it go via a particular city. The domain reasoner must first recognize how that constraint fits into the plan (a task shared with the dialogue modules, for example in determining which route is being modified). It then adds the constraint to the plan, possibly removing other softer constraints (such as to try an avoid cities known to be congested). It must then plan a new route satisfying the current constraints, preferably one that makes the smallest possible change to the plan already under consideration.

[1]The main bottleneck in TRAINS-95 is the speech recognition component. On an UltraSPARC with 192 MB of memory, recognition is effectively real-time.

Evaluation/comparison of options	25%
Suggesting courses of action	23%
Establishing world state	13%
Clarifying/confirming communication	13%
Discussing problem solving strategy	10%
Summarizing courses of action	10%
Identifying problems/alternatives	7%

Figure 3.1: Different forms of interaction in one hour of human-human problem-solving dialogues

3.1 Plan Representation and Reasoning

Although it's doing planning, a mixed-initiative planning system isn't doing what we might recognize as "traditional" planning, that is, constructing a sequence of operators from a fully-specified initial situation to a stated goal. In fact, in an informal analysis of one hour of human-human problem-solving dialogues (part of a larger eight hour study [Heeman and Allen, 1995], we found that a relatively small percentage of the utterances, 23%, dealt with explicitly adding or refining actions in the plan. A complete breakdown of the interactions is shown in Figure 3.1. Note the importance of being able to explicitly evaluate and compare options, even between humans of roughly equal ability. In human-computer interactions, we would expect this number to increase, as the computer can perform more and larger analyses. Similar conclusions about the nature of interactive planning are presented in [Ferguson, 1995] and [Pollack, 1992].

In [Ferguson *et al.*, 1996], we discuss why mixed-initiative planning seems to involve so little traditional planning. The main points focus on the fact that the initial situation, the goals, and the evaluation criteria for plans are all impractical to specify in complex domains, and this information is only acquired incrementally during the interaction. The upshot of this is that even if we had implemented a "perfect" route planner for the simple TRAINS-95 domain, we would still need all the other components of the system. If the goal is a natural, interactive planning assistant, the solution will not be found in traditional planning. The question becomes how to incorporate traditional systems within the context of a robust, mixed-initiative system.

We have developed a model of the mixed-initiative planning process from analysis of the TRAINS dialogues and implemented a simple version of it in TRAINS-95. This model consists of four steps:

1. Focus: Identify the goal/subgoal under consideration.
2. Gather Constraints: Collect constraints on the solution, selecting resources, gathering background information, and adding preferences.

3. Instantiate Solution: As soon as a solution can be generated efficiently, one is generated.

4. Criticize, Correct or Accept: If the instantiation is criticized and modifications are suggested, the process continues at step (2). If the solution appears acceptable then we continue at step (1) by selecting a new focus.

At first glance, this model seems quite similar to the expand-criticize cycle found in hierarchical planners since Sacerdoti [1977]. The significant difference is in step (3). Rather than pursuing a least commitment strategy and incremental top-down refinement, we "leap" to a solution as soon as possible. In TRAINS-95, solutions are generated by a domain-specific specialist program rather than by traditional search-based planning. We expect this will be typical in all practical applications of planning systems in the future. You might call this a "look then leap" strategy rather than the traditional "wait and see" strategy used in least-commitment planning.

3.2 TRAINS-95 Architectural Model

To support this model of mixed-initiative planning, we have developed a four layer architecture that generalizes the TRAINS-95 system architecture. The *discourse level* maintains information important for reference identification and for speech act interpretation and generation. While critical to the overall system, the other levels are more centrally concerned with the planning process.

The *problem solving level* maintains meta-level information about the problem solving tasks, similar to the problem solving level actions described by Litman and Allen [1987] and Lambert and Carberry [1991]. Actions at this level are problem solving actions such as "solve goal directly," "decompose goal into subproblems," "resolve resource conflict," and so on. This level supports processes such as identifying the task desired (*e.g.*, distinguishing between an elaboration of the plan for the current goal and shifting to another goal). It does this by maintaining an abstract tree of goals and information on which part of the problem is currently in focus. It also supports the process of determining the scope of problem solving actions such as cancellations and modifications. Finally, it supports discussion of the problem solving strategy to be used, and can maintain ordering constraints on when certain problem solving tasks should be performed.

When more sophisticated domain-level reasoning is required, we rely on the services of a set of *domain reasoners*. An *abstract plan representation level* manages the interface between the mixed-initiative system and these various domain specialists, matching open issues with reasoners and coordinating the responses. These reasoners might be, for example, a scheduler or an Internet agent that can retrieve information. The key to integrating such components is the ability to specify and reason about their capabilities. That is, the abstract plan reasoner needs to be able to reason about which domain reasoners might be suitable for what tasks, and interpret their results. This is complicated by the desire to use existing components

"off the shelf" as much as possible. The TRAINS system uses KQML [Finin *et al.*, 1993] to anticipate future integration efforts.

We have been concerned from the outset with the evaluation of our work, that is, how to know if we are making progress. This has traditionally been a problem for interactive systems and planners. Our current system has built into it a simple set of task-related metrics that are recorded for each interaction. As we refine those metrics, we can explore whether particular strategies are better than others at getting the task done, or whether the presence of certain components helps or hinders performance. Our first evaluation of the system is reported in [Allen *et al.*, 1996], where we show that most people can use the TRAINS system to solve 3-route problems effectively with virtually no training.

The TRAINS-95 system is a concrete first step towards a mixed-initiative planning assistant. We have demonstrated the feasibility of the dialogue-based approach to interactive planning, and have developed a substantial infrastructure for future research. More information on the TRAINS-95 system can be found in [Ferguson *et al.*, 1996] and [Allen *et al.*, 1996], and from our web site at URL:

`http://www.cs.rochester.edu/research/trains/`

4. Representing Action and Plans

A significant part of this project has focused on exploring common sense reasoning in the context of reasoning about actions and plans. The goal of the research is theories that account for human abilities to reason and communicate about their plans and actions, and systems based on those theories that can interact with people in a natural manner.

We are interested in *common sense theories*. This means, in particular, that techniques we develop must represent the world in a manner that is "natural" for people, which we take to mean that they can be described and discussed in natural language. Taking language descriptions seriously imposes quite strict breadth requirements on the theories. Of course, any theory must idealize certain aspects of the problem, and any computer program necessarily has its limits. But oversimplification is a problem that has led AI research down several unpromising (and worse, misleading) paths.

Reasoning about actions and plans has always been a core issue in artificial intelligence research. Indeed, the underlying questions of causality, intention, and physical intuitions have a rich intellectual history in fields ranging from philosophy to physics, although this history has not always been appreciated by researchers in AI. Our research addressed two main thrusts that span this spectrum.

4.1 Formal Foundations of Reasoning about Action and Time

The first was an exploration of the representation of actions in order to support natural, common sense reasoning. Traditional models of action in AI have been extremely weak, unable to represent such things as simultaneous actions or actions with durations. Of course, such phenomena are ubiquitous in realistic environments, and arise in any natural description of a scenario. Building on work by Allen [1984]), we re-examined the foundations of Interval Temporal Logic and its use in formalizing realistic domains naturally [Ferguson, 1992; Allen and Ferguson, 1994]. In this work we revisit the logical foundations of temporal logic, reexamining some of the very fundamental properties of the logic and discussing alternatives. Although quite technical, many of the points have clear natural analogs in terms of distinctions that people make in reasoning about action and time. We then move on to consider

actions and causality, and introduce *events* as an important part of the ontology. Events consist of something happening, as opposed to actions that are specific things that agents can do in the world, some of which result in events occurring. The connection between actions and events is context-dependent—an action doesn't always have the same effect.

One classical difficulty that besets planning (and reasoning about action in general) in any reasonably complex world is the Frame Problem, *i.e.*, the problem of succinctly characterizing and efficiently inferring what doesn't change when actions are performed. Schubert [1990] proposed a method of dealing simply with the Frame Problem (in non-probabilistic worlds) called *Explanation Closure* (EC). This method allows efficient, monotonic inference of non-change when considering a set of actions to be taken. During this project we explored this theory's potential and limitations for reasoning about actions and change. Using a fairly standard situation calculus representation, we tested it on the suite of problems collected by Sandewall [1992] for the purpose of examining the properties of various non monotonic approaches. We found that EC in combination with simple "action closure" (AC) axioms (stating that certain actions were the only ones taken within a certain setting) allowed simple and direct solution of all of Sandewall's problems, except for one probabilistic problem (a version of McCarthy's "potato in the tailpipe" problem). A probability-logic solution was proposed for this last problem, and it is expected that EC/AC methods can be generalized to probabilistic worlds. This work was reported as a contribution to a special issue on Actions and Processes of the *Journal of Logic and Computation* [Schubert, 1994].

We also explored the use of explanation closure with our Interval Temporal Logic representation. We again found that is well-suited to solving the Sandewall test suite problems, as well as being capable of handling a wide range of problems involving external events and simultaneous actions [Allen and Ferguson, 1994].

4.2 Reasoning about Plans

Another aspect of our work in this area was concerned with formal models of planning, or more generally, reasoning about plans. This obviously builds on the work on representing and reasoning about action, since plans fundamentally involve action. However, when we consider the many cognitive tasks that people perform with plans, especially in mixed-initiative planning, it becomes clear that plans involve more than just sequences of actions, contrary to the more-or-less standard approaches seen in AI planning formalisms.

Our early work on mixed-initiative planning [Ferguson and Allen, 1993b; Ferguson and Allen, 1993a], looked at approaches to plan reasoning that unified these tasks. Previous work on the various plan reasoning tasks had generally been disjoint, to the extent that the planning and plan recognition communities had little in common. When we started to take seriously the interactive nature of mixed-initiative planning, we saw that *plan communication* was the crucial task [Ferguson and Allen, 1994]. Informal statistics gathered from work on the TRAINS system (Figure 3.1 in Section 3) showed that many of the utterances were

concerned with keeping the conversation going: confirming, rejecting, clarifying, and so on. A relatively low proportion of the utterances were doing what might be recognizable as "AI planning," namely generating sequences of actions to satisfy a goal. We therefore proposed a formalism for representing plans which treats them as *arguments* in a formal system of defeasible reasoning (now called "computational dialectics"). This is an approach to nonmonotonic reasoning where arguments can be put forth to support some conclusion, drawing on the facts at hand as well as cases from previous experience. These arguments can be defeated by other arguments that attack their premises or methods, these attacks can themselves be defeated, and so on, until a conclusion is established that cannot be defeated (or all alternatives are exhausted). There are a variety of interesting technical details about argumentation as a form of inference (for example, it is clearly nonmonotonic, and in very interesting ways). But the main attraction of the model is its apparent similarity with the communicative "give-and-take" we see during mixed-initiative planning.

Ferguson's dissertation [1995] goes into more detail regarding argument-based reasoning, and looks at formalizing plans more concretely as arguments. That is, it is clear that we can allow arguments to be built from a knowledge base that contains the definitions of actions and events, causal rules, and so on. But what does this buy us? To begin with, the fact that arguments make explicit the assumptions they depend on allows us to connect them to an underlying logic of action and time. We also describe how the explanation closure approach used effectively in the representation of action plays a role in the formalization of plans as arguments. Finally, we look at recasting some existing planning formalisms, despite their weak expressivity, within the argument system approach, and show how some of the techniques and heuristics used traditionally appear as properties of the argument system.

4.3 Decision Theory, Simulation, and Planning

Another part of the project explored some applications of decision theory to plan reasoning. Developing a planning system that directly uses decision theory to generate plans is not to be successful because we are unlikely to have sufficiently detailed knowledge of the requisite probabilities and, if we did, the computations necessary to generate the an optimal plan would be overwhelming. Thus we concentrated on providing services that would extend the range of situations that a traditional planner could handle. These are:

- Determining the most likely (atomic) action to achieve a goal,
- Determining effective methods of executing plans abstract plans, and
- Monitoring the execution of plans and using this experience to improve the model.

To do this, we developed an event-based language for applying statistical tests to the problem of choosing appropriate actions when planning. Statistics are kept on the number of event types the agent encounters, and these statistics guide the choice of actions. Statistical

tests allow one discount probabilities for which one has only weak evidence. This work was presented at the International Conference on Statistics and Artificial Intelligence [Martin and Allen, 1993]. This knowledge representation language allows a planning system to reason about probability given its experience. By doing this, it links knowledge representation and machine learning. Programmers can encode their uncertain knowledge using the knowledge representation language described here, then, as the machine gathers experience, it can come to new conclusions about the uncertain knowledge the programmer encoded. Most current techniques are inadequate for this task because they do not distinguish carefully between the subjective probability the programmer encodes as the system's initial knowledge, and observations the system makes. Without such a distinction, connecting subjective probability and experience is difficult. The knowledge representation described here makes the necessary distinction and describes a technique for combining them.

This knowledge representation language is also unusual, if not unique, in that it uses interval estimates of probability. Interval estimates of probability are possible because we assume a frequentist definition of probability, another unusual feature of this knowledge representation language. Interval estimates of probability give a planner using the language the ability to reason about the strength of evidence it has for a probability.

In a related effort, we explored the use of simulation as a way of allowing a more-or-less traditional planner to handle mode complex situations, especially interacting simultaneous actions and external events. Rather than maintaining a STRIPS-style state-based representation to represent the changing situation, we used a coarse grained simulator that would simulate the execution of the plan based on an extended definition of the plan operators (with probabilities) and a model of the likelihood of various external events occurring in different situations. We use a fairly traditional hierarchical planer to generate possible plans which are then evaluated by a component that uses the results of many simulation runs to estimate the probabilities of various effects. Based on these results, the hierarchical planner can then extend the plan or backtrack in order to improve the probability of the desired effects. A preliminary report on this ongoing work is in [Yampratoom, 1994].

5. Temporal Reasoning Systems

The overall direction of our work in temporal reasoning during this project has been develop theoretical foundations and practical tools for scalable temporal reasoning and planning, keeping in mind the eventual needs in a transportation planning domain like TRAINS.

The results can be divided into theoretical development and practical implementation of extremely efficient methods for the "time point algebra" (relations using $<$, $\leq$, $=$, $\neq$) and theoretical complexity analysis of disjunctive temporal relations, and development of efficient methods for handling such disjunctions.

5.1 Scalable Temporal Reasoning

The starting point for this work was the TimeGraph I system [Miller and Schubert, 1990]. In experiments conducted by Yampratoom and Allen [1993], this system showed promise of being able to outperform all other existing temporal reasoning systems on large-scale TRAINS-world problems, involving qualitative time ordering information as well as numerical time constraints. The system uses a DAG representation of time point relations, with a superimposed metagraph structure that partitions the DAG into "time chains." Inference of implicit time-point relations is often near constant-time. Some weaknesses of TimeGraph I were the dependence of efficiency on the order in which temporal information is supplied, and inability to handle inequations of form $x \neq y$ for time points x and y. Also its reasoning is incomplete in the presence of numeric constraints.

Our first work was aimed initially at adding inequations to a qualitative time graph, *i.e.*, one without numerical information. (This means extending the convex point algebra to the full point algebra.) Our theoretical analysis uncovered a serious flaw in the proof of a lemma by van Beek and Cohen [1990], which is crucial to designing complete inference methods for time-point-algebra networks. We were able to formulate a completely new (and conceptually simpler) proof of the lemma [Gerevini and Schubert, 1993a; Gerevini and Schubert, 1995b], thus providing a sound basis for system-building. We further formulated efficient new algorithms for the "bottleneck" problem of explicitly deriving relations of form $x < y$ that are implicit in sets of relations of forms $x = y, w \neq z$.

We then proceeded to implement a new system, TimeGraph II, which improved on TimeGraph I in several respects: it builds a near-optimal TimeGraph structure for a given set

of temporal relations in linear time and space, also checking for consistency; it handles the full time-point algebra (but not numerical constraints); and it uses improved data structures and algorithms for graph construction and for inference, achieving a speed up of about a factor of 2 (for inference) over TimeGraph I. Furthermore, the implementation of the new algorithm for deriving implicit relations of form $x < y$ was shown experimentally to be orders of magnitude more efficient than the "minimal labels" algorithm that had been employed by van Beek [1992]. To our knowledge, TimeGraph II significantly outperforms all other comparable temporal reasoning systems. A theoretical description of TimeGraph II and experimental results are described in [Gerevini and Schubert, 1993b] (see also [Gerevini and Schubert, 1995a]). Descriptions of the use of TimeGraph I and II as temporal reasoning tools are given in [Gerevini *et al.*, 1993], and more fully [Gerevini *et al.*, 1995]. The version of TimeGraph II described there also includes methods for handling disjunctions (described below), and is available via anonymous ftp to `ftp.cs.rochester.edu`, in directory `pub/packages/knowledge-tools`, files `tg-ii-1.tar.gz` and `tg-ii.readme`.

5.2 Disjunctive temporal relations

The greatest source of computational complexity in qualitative temporal reasoning comes from disjunctive constraints such as $(x < y) \vee (w \leq z)$, since such constraints give rise to a combinatorial explosion in the pairwise orderings of time points that have to be considered in looking for a consistent solution. Yet such constraints are very important in planning and scheduling, since they are often necessary to ensure *disjointness* of intervals corresponding to events that must be scheduled in series (*e.g.*, because they demand a common resource), or *exclusion* of a point event from the interval between two other events (*e.g.*, where one of these events establishes a preconditions for the other, and insertion of the third event would "clobber" the precondition). These cases give rise to 3-point instances of disjunctive constraints, such as $(x < ymin) \vee (x > ymax)$.

Surprisingly, the complexity of such "point-interval exclusion" relations, as well as interval-interval exclusion, was not well understood, despite some relevant results on disjunctive relations on pairs of intervals by Golumbic and Shamir [1992], among others. This motivated a systematic theoretical investigation of point-based temporal disjointness, building on Golumbic and Shamir's work. We investigated the complexity of consistency-testing and solution-finding for 56 possible 3-point and 4-point relations (allowing for strictness and non-strictness of various ordering relations involved), and found that in the majority of cases these problems are NP-complete. The few polynomial cases are not very useful. (For instance, sets of interval-interval exclusion relations are trivially consistent when all ordering relations are non strict, merely because all given points can be consistently collapsed into a single time point). The strongest NP-completeness result concerns sets of very weak point-interval exclusion relations, of form "x is strictly before or strictly after the interval formed by y, z, where $y \neq z$ (but it is unspecified whether $y < z$ or $z < y$)". Preliminary results are reported

in [Gerevini and Schubert, 1993a] and the complete analysis is in [Gerevini and Schubert, 1994b].

These NP-completeness results led naturally to the next research issue: are there practical ways to deal with large sets of temporal relations that include disjunctions, despite the worst-case intractability of consistency testing (*etc.*) for such problems? Our interest was in complete methods, as opposed to polynomial-time but incomplete methods such as path consistency. The methods we developed were again based on TimeGraph structures, augmented with arbitrary sets of binary disjunctions of form $(x < y) \vee (w < z)$. The consistency-testing and solution-finding methods consist of a set of polynomial-time preprocessing techniques, followed by a form of intelligent backtrack search. As an example of a preprocessing step applied to $(x < y) \vee (w < z)$, the disjunct $(x < y)$ can be tested very quickly in the TimeGraph for truth or falsity. If it is true, the disjunction can be dropped; if it is false, the disjunct can be dropped. Of course in general the relations comprising the TimeGraph need not decide the truth or falsity of a disjunct, and search may be needed to determine which disjunct (if any) can consistently be made true.

Our very efficient preprocessing techniques turned out to have the interesting property that they are sufficient by themselves to determine consistency for Nebel and Buerckert's ORD-Horn algebra [Nebel and Buerckert, 1993], the maximal tractable subalgebra of Allen's interval algebra IA (among those subalgebras that include all the pointizable interval relations). Furthermore, our intelligent backtracking technique also turned out to be practically efficient, running in approximately quadratic time on average (relative to the number of given temporal relations, including disjunctive ones). These theoretical and experimental results are reported in [Gerevini and Schubert, 1994a; Gerevini and Schubert, 1995a].

In summary, our temporal reasoning work has led to complete, theoretically sound methods for qualitative reasoning about large sets of temporal relations that are extremely efficient in practice, allowing for fast inference of implicit relations, consistency testing, and solution-finding. Future work includes the design of methods that work well incrementally, i.e., when temporal relations may be added at any time; integration of qualitative relations with quantitative bounds (as in TimeGraph I, but with completeness guarantees); and generalization of disjunctive reasoning to handle a greater variety of disjunctions. With certain additions (including some ternary and quartic disjunctions) our methods would properly subsume IA, whereas at present they neither subsume IA nor are subsumed by it. However, our techniques as they stand and TimeGraph II seem well-suited to many problems, including the sorts of planning problems that provided our motivation.

6. Improving the Effectiveness of "Well-Founded" Planning

While there are some practically oriented planning systems such as SIPE [Wilkins, 1988] and O-Plan [Tate *et al.*, 1994] that could be applied to problems of modest size, these systems are quite complex. become ineffectual for large problems, and are logically incomplete and theoretically rather opaque. An ideal planner would be one that is simple and "well-founded" (*i.e.*, theoretically transparent, sound and complete), and scales up well to large problems. At present, however, well-founded planners such as UCPOP [Penberthy and Weld, 1992] are unable to solve any but trivially small problems. Our goal has been to find ways of accelerating such planners so as to bring them closer to practical usefulness.

Besides being applicable to the frame problem, Explanation Closure can also be used to infer actions that *must* be taken in any solution to a problem. Gerevini and Schubert explored the possibility of using such models to guide planners like UCPOP. It turned out that such planners are easily modified to emphasize addition of necessary actions, a strategy we term "zero commitment." As it happens, implementing zero-commitment in UCPOP does not require any technique as logically general as EC and AC, since UCPOP has a built-in STRIPS assumption, *i.e.*, any properties that are not explicitly declared to change in the definition of an action in fact remain unchanged when that action is instantiated. Thus, it was possible to experiment with the zero commitment strategy with only minor modifications to UCPOP; all that is required is a change to UCPOP's strategy for selecting "open conditions" (goals still to be established) so as to favor goals that can only be achieved by a unique action instance. Other goals are scheduled on a LIFO basis, and we term the resulting strategy ZLIFO.

ZLIFO turned out to be extremely effective, especially when combined with a modification of UCPOP's default A* strategy for selecting a plan to work on. The "mistake" in the default plan-selection strategy is to include a term reflecting the number of potential "clobbering" interactions between effects of actions and protected conditions (causal links). By eliminating or diminishing this term, we obtained large performance improvements in a wide spectrum of test problems from the UCPOP test suite, among others. Together, the new goal-selection and plan-selection strategies gave order of magnitude speedups, for all problems that were difficult for UCPOP to begin with. The hardest problems showed the greatest speedups (*e.g.*, from several minutes of CPU time to a fraction of a second, with similar improvements

in number of plans generated/ explored). The analysis and experiments are reported in [Schubert and Gerevini, 1995].

Finally, the most recent work on improving well-founded planning is concerned with the potential benefits of *preprocessing* operator and domain descriptions. The idea is to extract information that can be used to radically reduce the number of actions and states that need to be considered during planning. This holds great promise for making well-founded planning practical.

Our first effort in this direction was based on the observation that UCPOP generates many impossible actions when it performs goal regression, i.e., actions with parameter values that cannot possibly be instantiated, starting in the given initial conditions. We developed a preprocessing technique to calculate parameter domains (sets of constants) for all operators, based on "forward" propagation of constants. The idea is to match the initial conditions to all operator preconditions and thus associate potential values with some parameters in some operator preconditions. For those operators that had *all* preconditions matched, the potential values of the same parameter occurring in different preconditions can be intersected, and the effects of the operator can again be matched to operator preconditions, passing on the intersected value sets of the parameters; *etc.*. At the end all operator parameters have associated domains. What makes the algorithm nontrivial is the allowance in UCPOP for conditional effects in operator definitions, which may or may not lead to actual effects in particular uses of the operator.

A theoretical analysis of this technique showed that it runs in low-order polynomial time (in terms of the combined size of the operator specifications and the initial/goal state specifications). Our implementation gave negligible running times (relative to planning cost). We then modified UCPOP so that it eliminates actions with impossible parameter values (as determined by the precalculated domains), and also so that it eliminates apparent threats that would require impossible parameter values to be actualized. Our experimental tests concentrated on some of the harder problems from the UCPOP suite and on TRAINS world problems of the type actually used for some of the simplest TRAINS-91/93 dialogues. Typical speedups of a factor of 10 were obtained (beyond those obtained by our goal and plan selection strategies), bringing some simple problems within the realm of feasibility for UCPOP that previously could not be solved at all. A paper on this work has been accepted for conference presentation [Gerevini and Schubert, 1996].

In summary, our work on improving well-founded planning has led to order-of-magnitude speedups in state-of-the-art partial order planners, and further work should be able to make such planning practical. Current work on a preprocessing technique that infers state constraints from operator structure and initial conditions promises to provide another powerful means for cutting down search during planning. Another important future task is to modify our efficient temporal reasoning methods for use in planning. It should be possible to eliminate much of the searching in planning by using a TimeGraph-like reasoner to quickly find temporally consistent scenarios.

7. References

[Allen, 1984] James F. Allen, "Towards a general theory of action and time," *Artificial Intelligence*, 23:123–154, 1984. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 464–479.

[Allen and Ferguson, 1994] James F. Allen and George Ferguson, "Actions and events in interval temporal logic," *Journal of Logic and Computation*, 4(5):531–579, 1994. A much earlier version appeared in Working notes of the Second Symposium on Logical Formalizations of Commonsense Reasoning, Austin, TX, 11–13 January, 1993.

[Allen *et al.*, 1996] James F. Allen, Bradford W. Miller, Eric K. Ringger, and Teresa Sikorski, "A robust system for natural spoken dialogue," in *Proceedings of the Thirty-fourth Annual Meeting of the Association for Computational Linguistics (ACL-96)*, pages 62–70, 25–27 June 1996.

[Allen *et al.*, 1995] James F. Allen, Lenhart K. Schubert, George Ferguson, Peter Heeman, Chung Hee Hwang, Tsuneaki Kato, Marc Light, Nathaniel G. Martin, Bradford W. Miller, Massimo Poesio, and David R. Traum, "The TRAINS Project: A case study in defining a conversational planning agent," *Journal of Experimental and Theoretical AI*, 7:7–48, 1995. Also available as TRAINS Technical Note 93-4, Department of Computer Science, University of Rochester.

[Ferguson, 1992] George Ferguson, "Explicit Representation of Events, Actions, and Plans for Assumption-Based Plan Reasoning," Technical Report 428, Department of Computer Science, University of Rochester, Rochester, NY, June 1992.

[Ferguson, 1995] George Ferguson, *Knowledge Representation and Reasoning for Mixed-Initiative Planning*, PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, February 1995. Available as Technical Report 562, November, 1994.

[Ferguson *et al.*, 1996] George Ferguson, James Allen, and Brad Miller, "TRAINS-95: Towards a Mixed-Initiative Planning Assistant," in *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 70–77, Edinburgh, Scotland, 29–31 May 1996.

[Ferguson and Allen, 1993a] George Ferguson and James F. Allen, "Cooperative Plan Reasoning for Dialogue Systems," in *Papers from the AAAI-93 Fall Symposium on Human-Computer Collaboration: Reconciling Theory, Synthesizing Practice, AAAI Technical Report FS-93-05*, Raleigh, NC, 22–24 October 1993.

[Ferguson and Allen, 1993b] George Ferguson and James F. Allen, "Generic Plan Recognition for Dialogue Systems," in *Proceedings of the ARPA Workshop on Human Language Technology*, Princeton, NJ, 21–23 March 1993.

[Ferguson and Allen, 1994] George Ferguson and James F. Allen, "Arguing about Plans: Plan Representation and Reasoning for Mixed-Initiative Planning," in *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*, pages 43–48, Chicago, IL, 13–15 June 1994.

[Finin *et al.*, 1993] Tim Finin, Jay Weber, Gio Wiederhold, Michael Genesereth, Richard Fritzson, Donald McKay, James McGuire, Richard Pelavin, Stuart Shapiro, and Chris Beck, "Specification of the KQML Agent-Communication Language". Draft, 15 June 1993.

[Gerevini and Schubert, 1993a] Alphonso Gerevini and Lenhart K. Schubert, "Complexity of temporal reasoning with disjunctions of inequalities," in *Proceedings of the IJCAI-93 Workshop on Spatial and temporal Reasoning*, Chambery, France, 27August 1993.

[Gerevini and Schubert, 1993b] Alphonso Gerevini and Lenhart K. Schubert, "Efficient temporal reasoning through timegraphs," in Ruzena Bajcsy, editor, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 648–654, Chambery, France, 28 August–3 September 1993.

[Gerevini and Schubert, 1994a] Alphonso Gerevini and Lenhart K. Schubert, "An efficient method for managing disjunctions in qualitative temporal reasoning," in *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR-94)*, pages 214–2225, Bonn, Germany, 1994.

[Gerevini and Schubert, 1994b] Alphonso Gerevini and Lenhart K. Schubert, "On point-based temporal disjointedness," *Artificial Intelligence*, 70:347–361, 1994.

[Gerevini and Schubert, 1995a] Alphonso Gerevini and Lenhart K. Schubert, "Efficient algorithms for qualitative reasoning about time," *Artificial Intelligence*, 74(2):207–248, 1995.

[Gerevini and Schubert, 1995b] Alphonso Gerevini and Lenhart K. Schubert, "On computing the minimal labels in time point algebra networks," *Computational Intelligence*, 11(3):443–448, 1995.

[Gerevini and Schubert, 1996] Alphonso Gerevini and Lenhart K. Schubert, "Precomputation of parameter domains as an aid to planning," in *Proceedings of the Third Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 94–101, Edinburgh. Scotland, 29–31 May 1996.

[Gerevini *et al.*, 1993] Alphonso Gerevini, Lenhart K. Schubert, and Stephanie Schaeffer, "Temporal reasoning in TimeGraph I-II," *SIGART Bulletin*, 4(3):21–25, 1993.

[Gerevini *et al.*, 1995] Alphonso Gerevini, Lenhart K. Schubert, and Stephanie Schaeffer, "The temporal reasoning tools TimeGraph I-II," *International Journal of Artificial Intelligence Tools*, 4(1-2):281–299, 1995.

[Golumbic and Shamir, 1992] M. Golumbic and R. Shamir, "Algorithms and complexity for reasoning about time," in *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, 12–16 July 1992.

[Heeman and Allen, 1995] Peter A. Heeman and James F. Allen, "The TRAINS-93 Dialogues," TRAINS Technical Note 94-2, Dept. of Computer Science, University of Rochester, Rochester, NY, March 1995.

[Lambert and Carberry, 1991] Lynn Lambert and Sandra Carberry, "A tripartite plan-based model of dialogue," in *Proceedings of the Twenty-ninth Annual Meeting of the Association for Computational Linguistics (ACL-91)*, pages 47–54, Berkeley, CA, 18–21 June 1991.

[Litman and Allen, 1987] Diane Litman and James F. Allen, "A plan-recognition model for subdialogues in conversations," *Cognitive Science*, 11(2), 1987.

[Martin and Allen, 1993] Nathaniel G. Martin and James F. Allen, "Statistical Probabilities for Planning," in *Proceedings of the International Conference on Statistics and Artificial Intelligence*, Fort Lauderdale, FL, January, 1993. Also available as University of Rochester TR 474, November, 1993.

[Miller and Schubert, 1990] Stephanie Miller and Lenhart K. Schubert, "Time revisited," *Computational Intelligence*, 6:108–118, 1990.

[Nebel and Buerckert, 1993] Bernard Nebel and H. J. Buerckert, "Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra," *Journal of the Association for Computing Machinery*, 1993. To appear.

[Penberthy and Weld, 1992] J. Scott Penberthy and Daniel S. Weld, "UCPOP: A sound, complete, partial order planner for ADL," in Bernard Nebel, Charles Rich, and William Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR92)*, pages 103–114, Boston, MA, 25–29 October 1992. Morgan Kaufmann.

[Pollack, 1992] Martha E. Pollack, "The uses of plans," *Artificial Intelligence*, 57, 1992. Revised transcription of the Computers and Thought Award lecture delivered at the Twelth International Joint Conference on Artifical Intelligence (IJCAI-91), August 26, 1991, in Sydney, Australia.

[Sacerdoti, 1977] E.D. Sacerdoti, *A Structure for Plans and Behaviour*, Elsevier, North-Holland, New York, NY, 1977.

[Sandewall, 1992] Erik Sandewall, "Features and Fluents," Research report LiTH-IDA-R-92-30, Department of Computer and Information Science, Linköping University, S-581 83 Linköping, Sweden, September 1992. Second review version of parts of a forthcoming book.

[Schubert, 1990] Lenhart K. Schubert, "Monotonic Solution of The Frame Problem in The Situation Calculus: An Efficient Method for Worlds with Fully Specified Actions," in Henry E. Kyburg, Jr., Ronald P. Loui, and Greg N. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–68. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990. Also available as University of Rochester TR 306, August 1989.

[Schubert, 1994] Lenhart K. Schubert, "Explanation Closure, Action Closure, and the Sandewall Test Suite for Reasoning about Change," *Journal of Logic and Computation*, 4(5):679–799, 1994.

[Schubert and Gerevini, 1995] Lenhart K. Schubert and Alphonso Gerevini, "Accelerating partial order planners by improving plan and goal choices," in *Proceedings of the Seventh International Conference on Tools with AI (ICTAI-95)*, pages 442–450, 1995.

[Tate *et al.*, 1994] Austin Tate, Brian Drabble, and Richard Kirby, "O-Plan2: An open architecture for command, planning, and control," in M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*, pages 213–240. Morgan Kaufman, Palo Alto, CA, 1994.

[van Beek, 1992] Peter van Beek, "Reasoning about qualitative temporal information," *Artificial Intelligence*, 58(1-3):297–321, 1992.

[van Beek and Cohen, 1990] Peter van Beek and Robin Cohen, "Exact and approximate reasoning about temporal relations," *Computational Intelligence*, 6(3):132–144, August 1990.

[Wilkins, 1988] David E. Wilkins, *Practical Planning: Extending the Classical AI Planning Paradigm*, Morgan Kaufmann, San Mateo, CA, 1988.

[Yampratoom, 1994] Ed Yampratoom, "Using Simulation-based Projection to Plan in an Uncertain and Temporally Complex World," Technical Report 531, Department of Computer Science, University of Rochester, Rochester, NY, September 1994.

[Yampratoom and Allen, 1993] Ed Yampratoom and James F. Allen, "Performance of Temporal Reasoning Systems," *SIGART Bulletin*, 4(3):26–29, July 1993.

A. Bibliography of Publications Related to Rome Lab Contract F30602-91-C-0010, January 1991 – January 1996

Allen, J. F., "Planning as temporal reasoning," *Proc., 2nd Int'l. Conf. on Principles of Knowledge Representation and Reasoning*, 3–14, Cambridge, MA, April 1991.

Allen, J. F., "The RHET system," *Proc., AAAI Workshop on Implemented Knowledge Representation Systems*, March 1991.

Allen, J. F., "Time and time again: The many ways to represent time," *Int'l. J. of Intelligent Systems* 6(4), 341-356, July 1991.

Allen, J. F. and G. Ferguson, "Action in interval temporal logic," *Proc., 2nd Symp. on Logical Formalizations of Commonsense Reasoning*, U. Texas, Austin, January 1993.

Allen, J. F. and G. M. Ferguson, "Events and actions in interval temporal logic," *J. Logic and Computation* 4(5) (Special Issue on Actions and Processes), 531-579, Fall 1994; also appeared as "Actions and events in interval temporal logic," TR 521, Computer Science Dept., U. Rochester, July 1994.

Allen, J. F., G. M. Ferguson, B. W. Miller, and E. K. Ringger, "Spoken dialogue and interactive planning," *Proc., ARPA Spoken Language Technology Workshop*, Austin, TX, January 1995.

Allen, J. F., S. Hanks, P. Bonissone, P. Halverson, J. D. Tenenberg, and M. Vilain, "Strategies and cost in plan execution," RL-TR-91-431, Rome Laboratory, December 1991.

Allen, J. F. and B. W. Miller, "The RHET system: A sequence of self-guided tutorials," TR 325, Computer Science Dept., U. Rochester, July 1991.

Allen, J. F. and L. K. Schubert, "The TRAINS project," TR 382 and TRAINS TN 91-1, Computer Science Dept., U. Rochester, May 1991.

Allen, J. R., L. K. Schubert, G. M. Ferguson, P. A. Heeman, C-H. Hwang, T. Kato, M. N. Light, N. G. Martin, B. W. Miller, M. Poesio, and D. R. Traum, "The TRAINS project: A case study in defining a conversational planning agent," TRAINS TN 94-3 and TR 532, Computer Science Dept., U. Rochester, September 1994; *J. Experimental and Theoretical AI* 7, 7-48, 1995.

Bacchus, F., J. D. Tenenberg, and J. A. G. M. Koomen, "A non-reified temporal logic," *Artificial Intelligence* 52(1), November 1991.

Ferguson, G. M., "Domain plan reasoning in TRAINS-90," TRAINS TN 91-2, Computer Science Dept., U. Rochester, June 1991.

Ferguson, G. M., "Explicit representation of events, actions, and plans for assumption-based plan reasoning," TR 428 and Thesis Proposal, Computer Science Dept., U. Rochester, June 1992.

Ferguson, G. M., "Knowledge representation and reasoning for mixed-initiative planning," TR 562 and Ph.D. Thesis, Computer Science Dept., U. Rochester, January 1995.

Ferguson, G. M. and J. F. Allen, "Arguing about plans: Plan representation and reasoning for mixed-initiative planning," *Proc., 2nd Int'l. Conf. on AI Planning Systems*, 43–48, Chicago, IL, June 1994.

Ferguson, G. M. and J. F. Allen, "Cooperative plan reasoning for dialogue systems," position paper, *AAAI Fall Symp. on Human-Computer Collaboration*, Raleigh, NC, October 1993.

Ferguson, G. and J. F. Allen, "Generic plan recognition for dialogue systems," *ARPA Workshop on Human Language Technology*, Princeton, NJ, March 1993.

Ferguson, G. M., J. F. Allen, and B. W. Miller, "TRAINS-95: Towards a mixed-initiative planning assistant," to appear, *Proc., 3rd Conf. on Artificial Intelligence Planning Systems (AIPS-96)*, Edinburgh, Scotland, May 1996.

Gerevini, A. and L. K. Schubert, "Complexity of temporal reasoning with disjunctions of inequalities," *Workshop on Spatial and Temporal Reasoning, 13th Int'l. Joint Conf. on Artificial Intelligence*, Chambéry, France, August 1993.

Gerevini, A. and L. K. Schubert, "Efficient algorithms for qualitative reasoning about time," TR 496, Computer Science Dept., U. Rochester, March 1994; *Artificial Intelligence* 74(2), 207-248, 1995.

Gerevini, A. and L. K. Schubert, "An efficient method for managing disjunctions in qualitative temporal reasoning," *Proc., Conf. on Principles of Knowledge Representation and Reasoning (KR-94)*, 214-225, Bonn, Germany, May 1994; *Artificial Intelligence* 70, 1994; preliminary version in *Workshop Notes of the ECAI-94 Workshop on Algorithms, Complexity, and Commonsense Reasoning*, Amsterdam, August 1994.

Gerevini, A., and L. K. Schubert, "Efficient temporal reasoning in Timegraph II," presented, *Italian Planning Workshop (IPW '93) (Workshop Italiano sulla Pianificazione Automatica 1993)*, Rome. Italy, September 1993.

Gerevini, A. and L. K. Schubert, "Efficient temporal reasoning through timegraphs," *Proc., 13th Int'l. Joint Conf. on Artificial Intelligence (IJCAI-93)*, 648-654, Chambéry, France, August-September 1993.

Gerevini, A., and L. K. Schubert, "On computing the minimal labels in time point algebra networks," TR 495, Computer Science Dept., U. Rochester, March 1994; *Computational Intelligence* 11(3), 443-448, August 1995.

Gerevini, A., and L. K. Schubert, "On point-based temporal disjointness," TR 497, Computer Science Dept., U. Rochester, March 1994; *Artificial Intelligence* 70, 347-361, October 1994; preliminary version in *Workshop Notes of the ECAI-94 Workshop on Algorithms, Complexity, and Commonsense Reasoning*, Amsterdam, August 1994.

Gerevini, A. and L. K. Schubert, "Precomputation of parameter domains as an aid to planning," to appear. *Proc., 3rd Int'l. Conf. on Planning Systems (AIPS-96)*, Edinburgh, Scotland, May 1996.

Gerevini, A., L. K. Schubert, and S. Schaeffer, "Temporal reasoning in Timegraph I-II," *SIGART Bulletin* 4(3), 21-25, July 1993.

Gerevini, A., L. K. Schubert, and S. Schaeffer, "The temporal reasoning systems Timegraph I-II," TR 494, Computer Science Dept., U. Rochester, March 1994; revised April 1994.

Gerevini, A., L. K. Schubert, and S. Schaeffer, "The temporal reasoning tools Timegraph-I-II," *Proc., 6th IEEE Int'l. Conf. on Tools with Artificial Intelligence*, New Orleans, LA, November 1994; *Int'l. J. Artificial Intelligence Tools* 4(1-2), 281-299, 1995.

Gross, D., J. F. Allen, and D. R. Traum, "The TRAINS 91 dialogues," TRAINS TN 92-1, Computer Science Dept., U. Rochester, June 1993.

Hoebel, L. J., "Anytime spatio-temporal constraint propagation," *Working Notes, AAAI 1994 Workshop on Spatial and Temporal Reasoning*, Seattle, WA, July 1994.

Hoebel, L. J., "Combining qualitative and quantitative spatial and temporal information in a hierarchical structure," *6th Annual Workshop on Space Applications and Research (SOAR-92)*, 265-274, NASA Conf. Publication 3187, August 1992.

Hoebel, L. J., "Tractable anytime temporal constraint propagation," abstract, *Proc., 12th Nat'l. Conf. on Artificial Intelligence (AAAI-94)*, Seattle, WA, August 1994.

Hwang, C-H. and L. K. Schubert, "EL: A formal, yet natural, comprehensive knowledge representation," *Proc., 11th Nat'l. Conf. on Artificial Intelligence (AAAI-93)*, 676-682, Washington, DC. July 1993.

Hwang, C-H. and L. K. Schubert, "EL: A representation that lets you say it all," in N. Guarino and R. Poli (Eds.). *Preliminary Proc., Int'l. Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation*, 277-289, Padova, Italy, March 1993.

Karlsson, J., "Task decomposition in reinforcement learning," *Proc., AAAI Spring Symposium on Goal-Driven Learning*, Palo Alto, CA, March 1994.

Martin, N. G., "Applying statistical inference to planning under uncertainty," *SIGART Bulletin*, October 1993; forthcoming TR 472 and Ph.D. Thesis, Computer Science Dept., U. Rochester, expected Spring 1995.

Martin, N. G. and J. F. Allen, "A decision theoretic planning assistant," *AAAI Spring Symp. on Active Natural Language Processing*, Palo Alto, CA, March 1994; *ARPA / Rome Labs Planning Initiative Workshop*, Tucson, AZ, February 1994.

Martin, N. G. and J. F. Allen, "A language for planning with statistics," *Proc., Uncertainty in Artificial Intelligence* Anaheim, CA, Morgan Kaufmann, July 1991.

Martin, N. G. and J. F. Allen, "Statistical probabilities for planning," presented, *Int'l. Conf. on Statistics and Artificial Intelligence*, Ft. Lauderdale, FL, January 1993; TR 474, Computer Science Dept., U. Rochester, November 1993.

Martin, N. G. and B.W. Miller, "The TRAINS-90 simulator," TRAINS TN 91-4, Computer Science Dept., U. Rochester, May 1991.

Martin, N. G. and G. J. Mitchell, "TRAINS world simulator: User's manual," TRAINS TN 95-2, Computer Science Dept., U. Rochester, September 1995.

Martin, N. G. and G. Mitchell, "A transportation domain simulation for debugging plans," *Proc., IEEE Dual Use Conference*, Utica, NY, May 1994.

Murtezaoglu, B., "Investigation of evidence combination methods in evidential probability," TR 391, Computer Science Dept., U. Rochester, revised December 1992.

Murtezaoglu, B. and H.E. Kyburg, Jr., "A modification to evidential probability," TR 378, Computer Science Dept., U. Rochester, July 1991; *Proc., Uncertainty in Artificial Intelligence*, 228-231, Anaheim, CA, July 1991.

Poesio, M., G. M. Ferguson, P.A. Heeman, C-H. Hwang, D. R. Traum, J. F. Allen, N. G. Martin, and L. K. Schubert, "Knowledge representation in the TRAINS system," *AAAI 1994 Fall Symp. on Knowledge Representation for Natural Language Processing in Implemented Systems*, New Orleans, LA, November 1994.

Schubert, L. K., "Explanature closure, action closure, and the Sandewall test suite for reasoning about change," TR 440, Computer Science Dept., U. Rochester, October 1992; revised June 1994; *J. Logic and Computation* 4(5) (Special Issue on Actions and Processes), 679-700, 1994.

Schubert, L. K. and A. Gerevini, "Accelerating partial order planners by improving plan and goal choices," TR 570, Computer Science Dept., U. Rochester, January 1995; revised April 1995; *Proc., 7th Int'l. Conf. on Tools with AI (ICTAI -95)*, 442-450, Washington, DC, November 1995; TR 607, Computer Science Dept., U. Rochester, January 1996; expanded and revised version submitted for journal publication.

Tenenberg, J. D., "Abandoning the completeness assumptions: A statistical approach to the frame problem," *Int'l. J. of Expert Systems* 3(3), 1991; also in K. Ford and P. Hayes (Eds.). *Advances in Human and Machine Cognition I: The Frame Problem in Artificial Intelligence.* 1991.

Tenenberg, J. D., J. Karlsson, and S. D. Whitehead, "Learning via task decomposition." *Proc., 2nd Int'l. Conf. on Simulation of Adaptive Behavior: From Animals to Animats* (Honolulu, HI, December 1992), MIT Press, 1993.

Traum, D. R. and J. F. Allen, "Causative forces in multi-agent planning," *Proc., 2nd European Workshop on Modeling Autonomous Agents and Multi-Agent Worlds (MAA-MAW90)*, 135-139, France, August 1990; also in Y. Demazeau and J.-P. Müller (Eds.). Decentralized Artificial Intelligence II. Elsevier Science Publishers B.V., 89-105, 1991.

Traum, D. R. and J. F. Allen, "Towards a formal theory of repair in plan execution and plan recognition," *13th Workshop of the UK Planning and Scheduling Special Interest Group*, Glasgow, Scotland, September 1994.

Traum, D. R., J. F. Allen, G. M. Ferguson, P.A. Heeman, C-H. Hwang, T. Kato, N. G. Martin, M. Poesio, and L. K. Schubert, "Integrating natural language understanding and plan reasoning in the TRAINS-93 conversational system," *Working Notes: AAAI Spring Symp. Series: Active Natural Language Processing*, Palo Alto, CA, March 1994.

Traum, D. R., L. K. Schubert, M. Poesio, N. G. Martin, M. N. Light, C-H. Hwang. P. A. Heeman, G. M. Ferguson, and J. F. Allen, "Knowledge representation in the TRAINS-93 conversation systems," *Int'l. J. Expert Systems* (Special Issue on Knowledge Representation and Inference for Natural Language Processing), to appear, 1996.

Whitehead, S. D., J. Karlsson, and J. D. Tenenberg, "Learning multiple goal behavior via task decomposition and dynamic policy merging," in S. Mahadevan and J. Connell (Eds.). *Robot Learning.* Cambridge, MA: MIT Press, 1993.

Yampratoom, E., "Using simulation-based projection to plan in an uncertain and temporally complex world," TR 531, Computer Science Dept., U. Rochester, September 1994.

Yampratoom, E. and J. F. Allen, "Performance of temporal reasoning systems," TRAINS TN 93-1, Computer Science Dept., U. Rochester, March 1993; revised May 1993: *SIGART Bulletin* 4(3), 26-29, July 1993.

Appendix B

TRAINS-95: Towards a Mixed-Initiative Planning Assistant*

Abstract

We have been examining mixed-initiative planning systems in the context of command and control or logistical overview situations. In such environments, the human and the computer must work together in a very tightly coupled way to solve problems that neither alone could manage. In this paper, we describe our implementation of a prototype version of such a system, TRAINS-95, which helps a manager solve routing problems in a simple transportation domain. Interestingly perhaps, traditional planning technology does not play a major role in the system, and in fact it is difficult to see how such components might fit into a mixed-initiative system. We describe some of these issues, and present our agenda for future research into mixed-initiative plan reasoning. At this writing, the TRAINS-95 system has been used by more than 100 people to solve simple problems at various conferences and workshops, and in our experiments.

1 Introduction

In command and control situations and logistics planning, a human planner faces several difficult problems. First, there is a surplus of data, only a small amount of which is actually information relevant to the current task. In fact, what data is relevant cannot be determined in advance and only becomes clear as the situation and the plan develop. Second, the plans being considered are large and complex, and it is beyond human capabilities to manage all the details effectively. Automated planning systems are better able in principle to handle the

*This material is based on: George Ferguson, James Allen, and Brad Miller, "TRAINS-95: Towards a Mixed-Initiative Planning Assistant," *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pp. 70–77, Edinburgh, Scotland, 29–31 May, 1996.

scale, but are hard to apply because of the under-specified initial situations, and the fact that many planning decisions are made on an intuitive basis that cannot be effectively quantified. As a result, neither the human or the computer can effectively solve such planning problems in isolation. This problem motivates an interest in mixed-initiative planning systems, where the computer acts as a collaborating assistant to the human, anticipating needs, performing the tasks it is well suited for, and leaving the remaining tasks to the human. By cooperating, the human-computer "team" may be able to effectively deal with problems that neither could handle alone.

To explore these aspects of mixed-initiative planning, we have built a prototype system, TRAINS-95, that helps a manager solve routing problems in a simple transportation domain. The manager is presented with a map displaying cities and rail connections between them, as shown in Figure 1. The system generates random problems that require planning routes for a set of engines to a set of destinations. Various environmental factors can arise during the interaction, which the manager and system must then plan to accommodate. The system can interact using both spoken and typed English, as well as graphical displays and controls.

Our goal was a robust, modular, multi-modal, mixed-initiative planning assistant. To be more specific, by "robustness" we mean that no matter what occurs during the interaction, the system not only doesn't crash, but does something to indicate its understanding of the manager's intentions and its own attempts to further the plan. By "modular" we are taking seriously the idea that there are, or will be shortly, a variety of knowledge sources, reasoning agents, and display engines available as resources that the system can employ. Examples include weather information, newsfeeds, map servers, and so on, as well as "off-the-shelf" technology such as speech recognizers and generators. As we will describe, many of the components of TRAINS-95 are themselves autonomous modules connected by a general-purpose message-passing process manager.

By "multi-modal" we mean that there are a multitude of ways of communicating between humans and computers, include speech input and output, written text, and graphical displays such as maps, charts, forms, and the like, for both input and output. From the outset, we intended that TRAINS-95 should accommodate all of these, allowing the human to interact in whatever way seems most natural to them. In some situations existing procedures may impose constraints on what form communication can take (*e.g.*, an operator may be trained on a particular type of display). In other cases determining the appropriate modality is an ongoing process, and the subject of current research. By treating all modalities as *linguistic*, that is, as a form of language, we obtain a powerful unifying model of the interaction as a form of *dialogue*. Of course, multi-modal communication raises a new set of issues for all aspects of the system, as we will discuss later.

Finally, by "mixed-initiative" we mean that both the system and the human are on roughly equal ground as participants in the dialogue. This is not to say that they are equals, since clearly they are good at different things. But in a truly mixed-initiative system, both participants can do what they do best. The human typically has knowledge of the high-level goals and means of achieving them, while of course, the computer is good at managing the

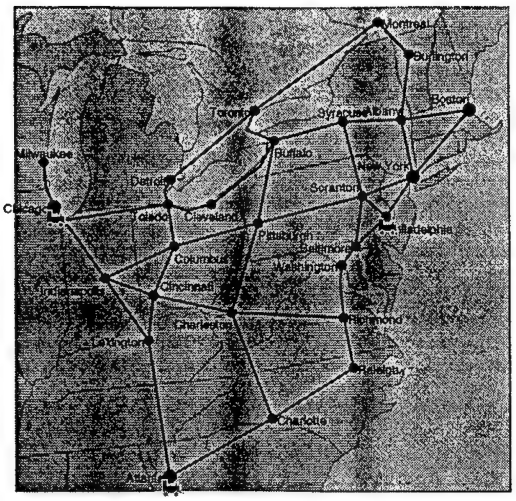


Figure 1: TRAINS-95 System Map Display

multitude of low-level details that go into such plans. As well, in realistic situations, the human may be guided by principles that are difficult or impossible to quantify precisely, making it impossible to fully automate the task and replace them. Even if this were possible in principle, the resulting master-slave style of communication would hardly be natural and probably wouldn't be very efficient.

The next section describes our current implementation, TRAINS-95, including design goals, our approach to the aspects outlined above, and lessons learned. The key points are: (1) that dialogue provides the context required for successful interaction independent of input modality; and (2) that the plan reasoning requirements of mixed-initiative systems differ markedly from the specifications of traditional planners. We elaborate on these points in the following section. In the final section we present some of the many research issues raised by our work on TRAINS-95, and describe the directions we are pursuing in addressing them.

2 The TRAINS-95 System

The TRAINS-95 system is implemented as a set of independent modules, as shown in Figure 2. The modules shown in the dashed box are implemented within a single Lisp program, otherwise each module is a separate Unix process, and modules communicate by exchanging messages. We have developed a general-purpose Unix process manager that manages the various processes and their communication. It performs the mapping between logical module names and physical processes, and provides services such as broadcast messages and complete logging and replay of message exchanges. It is itself controlled by messages, for

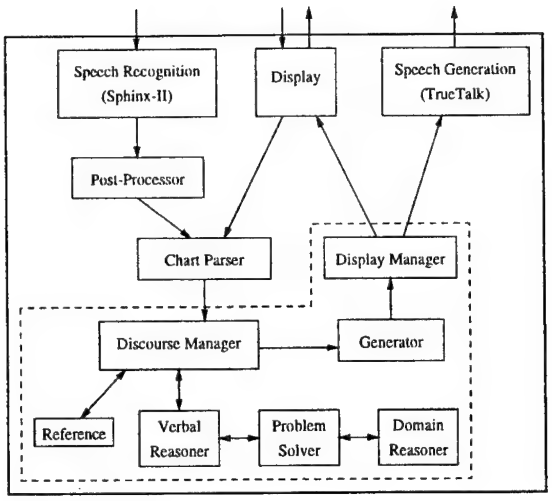


Figure 2: TRAINS-95 System Architecture

example to create a new process, restart or kill a module, register interest in another module, and so on.

In building the current system, we have concentrated on the interface and infrastructure components, in order to get a system that naive users could interact with as soon as possible. In this section we will describe the various modules and their interactions, concentrating on those aspects central to the goal of a robust, multi-modal application. The next section provides an example of the system in action and discusses the implications for mixed-initiative planning systems.

2.1 Speech and Display Modules

For speech input in TRAINS-95, we use the Sphinx-II recognizer from CMU [Huang *et al.*, 1992], trained on the ATIS airline reservation corpus, with a vocabulary of approximately 3000 words. We deliberately did not build a special recognizer for our task or domain, since we wanted to investigate using off-the-shelf components directly as "black boxes" in our system. Of course, this adversely impacts the rest of the system, since many of the words in the ATIS corpus are meaningless in the simple transportation domain we are dealing with in TRAINS-95 (*e.g.*, "DC3," "Riyadh," and "movie"). We considered this an appropriate test of robustness, and something which any system that intends to use such "plug-and-play" components must ultimately address. In this particular case, the output of the module is simple enough, namely a sequence of words (and control messages for when a hypothesis needs to be revised), that the semantics of its messages is obvious.

To compensate for the under-constrained speech recognizer, its output is post-processed by a domain- and/or speaker-dependent module [Ringger, 1995]. This module uses tech-

niques from machine translation to "correct" the output of the recognizer, based on a statistical model derived from a corpus of previous runs of the system.

The speech generation component of TRAINS-95 is built on the TrueTalk generator from Entropics, Inc. This program provides high-quality speech generation and a wide range of intonational and inflectional controls.

Although speech is the most exciting and challenging modality, other modalities are provided by an object-oriented display module. For input, it allows typed input, menu selections, and direct mousing on display elements, all of which are broadcast to other modules. Output to the display is performed using a fairly rich object-oriented language, allowing a variety of graphical elements to be defined and displayed. The current version of the display supports object types such as "engine," "track," and "region;" in the immediate future this will be generalized even further. Several types of dialog box can be popped up (under the control of the language components), allowing yet another input modality.

2.2 Language and Dialogue Processing

The core of the TRAINS-95 system is the set of modules that perform language understanding and dialogue processing tasks. The first of these is a robust parsing system based on a bottom-up chart parser [Allen, 1994]. The parser uses a fairly comprehensive lexicon of 1700 words and grammar of 275 rules, both built from the TRAINS dialogue corpus [Heeman and Allen, 1995]. The parser accepts input from all the input modules—we treat spoken, typed, and moused input uniformly as linguistic communication. The parser's output is a set of *speech acts* expressing the content of the utterance. Since the majority of utterances in spoken dialogue are not complete syntactic sentences, the traditional chart parser is augmented with a set of *monitors*. If a complete parse is not found, these monitors scan the chart and attempt to piece together a partial representation of what was said. This approach to natural language understanding is crucial to meeting the goal of a robust system. It also means that the system can often interact effectively even at very low speech recognition rates.

After the parser has determined a set of possible speech acts for the utterance, the reference module deals with grounding any terms to the objects in the domain (*e.g.*, "to Chicago, " "send it"), or turning indefinite references into a query to be resolved later (*e.g.*, "an engine").

The verbal reasoner is then responsible for interpreting the speech acts in context, querying various knowledge sources in order to understand or address the speech act, formulating responses, and maintaining the system's idea of the state of the discourse. The verbal reasoner architecture is based on a set of prioritized rules that match patterns in the input speech acts. These rules again allow robust processing the face of partial or ill-formed input. The presence of an element such as "from Avon" may allow it to determine that the user is requesting a plan incorporation. These rules rely heavily on the *discourse context*, that collection of knowledge and beliefs that describe the state of the interaction. In fact. it

is precisely this context that makes dialogue-based interaction so much more flexible and powerful than one-shot transaction-based communication.

Speech acts that appear to be requests to incorporate a constraint on a prior system action or to extend the current plan are passed on to the problem solver. This module is also based on a prioritized set of rules, this time for dealing with user responses to system questions or with plan corrections and extensions. These rules turn the information into queries and updates to a domain reasoner, *e.g.*, for a path that fits the constraints "from Chicago to New York via Pittsburgh." If fragmentary information is supplied from the verbal reasoner (which will generally try to fill it out with information from the discourse context), then the problem solver attempts to incorporate the fragment into what it knows about the current state of the plan. For instance, if the prior action had been to propose a route, and the verbal reasoner did not indicate in its request to the problem solver that the current request is part of a rejection, then it will attempt to select another goal the user may have informed it of, and incorporate the fragment with respect to that goal (presuming the fragment does not contain information that indicates a goal shift).

The output of the verbal reasoner is a set of speech acts that the system wants to communicate to the user. The generation module decides how to actually achieve the communication using either the display or the speech generator, or both. This module also uses prioritized rules to match requested speech acts to appropriate means of expressing them. For instance, a request to the generator to warn the user of track repair along the route between Chicago and Toledo will result in telling the display to flash the track segment red, and simultaneously speak the phrase "the track between Chicago and Toledo is currently undergoing repair." The generation module (along with reference) maintains a representation of what the system believes the user already knows, to prevent overly verbose or confusing output.

The verbal reasoning components of TRAINS-95 attempt to provide a robust agent that keeps the dialogue going. The approach is generally to instantiate the current plan as soon as possible, communicate the results, and allow the user to correct or elaborate it. This "look then leap" strategy will be further described in the next section.

2.3 Domain Reasoning

The domain reasoner in TRAINS-95 maintains a knowledge base describing the state of the world and provides planning and plan recognition services to the language modules. For the simple route-planning domain, of course, it would be easy to build a perfect reasoner that solved the problems as soon as the manager had stated their goals. However, not only is it unlikely that we will ever be able to build such a reasoner for a realistic domain, in the next section we claim that such a system is not necessarily appropriate for mixed-initiative planning. We therefore deliberately weakened the TRAINS-95 domain reasoner in order to force the manager to interact in order to overcome its shortcomings. The route planner can therefore only plan route segments less than four hops long. and for those it chooses a random path. The knowledge base maintains an accurate view of the map, and allows various

"natural" events such as bad weather or track maintenance to arise during the interaction. These also force interaction in order to revise plans to take account of them.

An important aspect of the type of domain reasoning that is performed in TRAINS-95 is that it is highly incremental and incorporates aspects of both planning and plan recognition in a tightly-coupled way. For example, the domain reasoner may be asked to incorporate a new constraint on an existing plan, *e.g.*, that it go via a particular city. The domain reasoner must first recognize how that constraint fits into the plan (a task shared with the language modules, for example in determining which route is being modified). It then adds the constraint to the plan, possibly removing other softer constraints (such as to try an avoid cities known to be congested). It must then plan a new route satisfying the current constraints, preferably one that makes the smallest possible change to the plan already under consideration.

This model, it seems to us, is quite unlike the traditional view of planning, where a goal is specified (completely) and the planner goes off and produces a plan to achieve it. This raises a question as to the role of traditional planning in mixed-initiative systems. The next section elaborates on this point.

3 Where's the Planning in Mixed-Initiative Planning?

Figure 3 presents a example of an actual dialogue between a user and the TRAINS-95 system, starting from the situation shown in Figure 1. The task is admittedly very simple and, as we noted previously, the system's planning abilities were deliberately limited in order to encourage interaction. The point of the dialogue is to emphasize that this is indeed a mixed-initiative *planning* system. The user and the system are constructing, evaluating, and correcting plans, albeit simple ones. The environment is dynamic, with problems such as weather and traffic occurring independent of the planner. The human manager's goals and preferences need not be explicit, as when they choose to ignore the problem at Toledo. The system has clear limitations, as it would in any realistic domain, and it understands how to deal with these by interacting with the manager.

Although it's doing planning, however, a mixed-initiative planning system isn't doing what we might recognize as "traditional" planning, that is, constructing a sequence of operators from a fully-specified initial situation to a stated goal. In fact, in an informal analysis of one hour of human-human problem-solving dialogues (part of a larger eight hour study [Heeman and Allen, 1995]), we found that a relatively small percentage of the utterances, 23%, dealt with explicitly adding or refining actions in the plan. Figure 4 summarizes this analysis. Note the importance of being able to explicitly evaluate and compare options, even between humans of roughly equal ability. In human-computer interactions, we would expect this number to increase, as the computer can perform more and larger analyses. Similarly, even in the very simple TRAINS world, a significant portion of the communication was concerned with establishing the state of

```
1.  M: I need to get to Detroit, Syracuse, and Montreal.
2.  S: Ok. [highlights the three cities]
3.  M: Let's send the trains from Chicago to Detroit.
4.  S: Ok. [draws route from Chicago to Detroit via Toledo]
       But the terminal at Toledo is delaying traffic due to bad weather.
       Trains will take an additional five hours to move through there.
       [highlight Toledo red]
5.  M: That's ok. Now let's go from Atlanta up to Syracuse.
6.  S: Please tell me a route to use to get from Atlanta to Syracuse.
       [highlight engine at Atlanta and city Syracuse]
7.  M: Go via Charleston and Pittsburgh.
8.  S: Ok. [draws route via Cincinnati]
       But traffic through Cincinnati is being delayed due to construction.
9.  M: Ok, let's go via Charleston and Buffalo instead.
10. S: No problem. [draws route]
11. M: That's great.
12. S: I hope it was helpful.
13. M: Finally, I want to go to Montreal from Philadelphia.
14. S: Ok. [draws route]
15. M: That's great. I'm done.
```

Figure 3: Actual TRAINS-95 Dialogue

Evaluation/comparison of options	25%
Suggesting courses of action	23%
Establishing world state	13%
Clarifying/confirming communication	13%
Discussing problem solving strategy	10%
Summarizing courses of action	10%
Identifying problems/alternatives	7%

Figure 4: Analysis of one hour of human-human problem-solving dialogues

the world, and we would expect this to increase in a more realistic environment. Similar conclusions about the nature of interactive planning are presented in [Ferguson, 1995; Pollack, 1992].

Why is it that mixed-initiative planning seems to involve so little traditional planning (finding courses of action from initial situation to goal)? In the first place, it is because the initial situation is usually incompletely specified, whether because of changing conditions as in our simple example or in a more realistic setting because it is simply too huge to represent. Rather, the determination of what aspects of the world are relevant to the plan is a dynamic process that is intimately connected with the exploration and elaboration of the plan. It would be impossible in practice to extract such criteria from the human manager offline.

Similarly, the goals of the plan in the mixed-initiative setting are also poorly specified. Not only do they change over time as the manager's preferences and concerns change, but they typically cannot be extracted and that knowledge codified into the system's operation. Of course, in our simple example the only goal the manager can have is to get trains to their destinations, so the system can make simplifying assumptions. But even so, treating human-imposed constraints on the plan as part of the "goals," the reasons for rejecting a particular plan may not be made explicit to the system.

The upshot of this is that even if we had implemented a "perfect" route planner for the simple TRAINS-95 domain, we would still need all the other components of the system. If the goal is a natural. interactive planning assistant, the solution will not be found in the language of traditional planning. The question becomes how to incorporate traditional systems that are good at what they do but relatively inflexible and brittle within the context of a robust, mixed-initiative system.

We have developed a model of the mixed-initiative planning process from analysis of the TRAINS dialogues and implemented a simple version of it in TRAINS-95. This model consists of four steps:

A. Focus: Identify the goal/subgoal under consideration.

B. Gather Constraints: Collect constraints on the solution, selecting resources, gathering background information, and adding preferences.

C. Instantiate Solution: As soon as a solution can be generated efficiently, one is generated.

D. Criticize, Correct or Accept: If the instantiation is criticized and modifications are suggested, the process continues at step (B). If the solution appears acceptable (given the current focus), then we continue at step (A) by selecting a new focus.

At first glance, this model seems quite similar to the expand-criticize cycle found in hierarchical planners since Sacerdoti [Sacerdoti, 1977]. The significant difference is in step (C). Rather than pursuing a least commitment strategy and incremental top-down refinement, we "leap" to a solution as soon as possible. You might call this a "look then leap" strategy

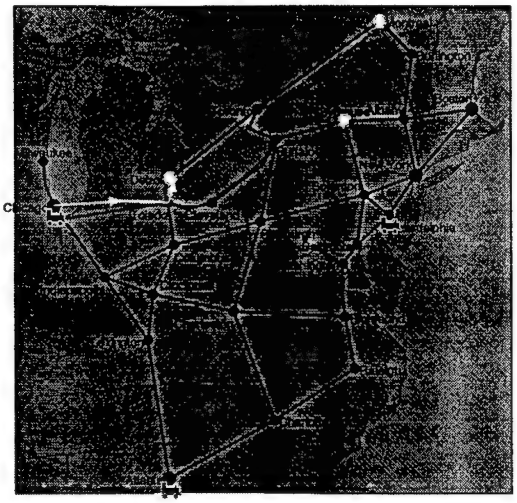


Figure 5: TRAINS-95 System Map Display after (4)

rather than the traditional "wait and see" strategy used in least-commitment planning. As such, our strategy shares similarities with Sussman's original work with an emphasis on plan debugging and repair [Sussman, 1974]. When the plan must be a result of a collaboration, it seems that it is more efficient to throw out a solution and criticize it, rather than to work abstractly as long as possible. Notice that this is exactly why committees often start with a "straw man" when trying to develop new policies. This helps the committee focus its discussion, just as the the plans under discussion in mixed-initiative planning provide part of the context within which the manager is exploring the problem.

To see how the "look then leap" model fits well with the discourse strategies used in our dialogues, consider the simple TRAINS-95 dialogue in Figure 3 again. In (1), the manager clearly sets out a set of goals, limiting but not fully specifying the focus (step A). Statement (3) implicitly focuses on the goal of getting to Detroit, and starts adding constraints (step B). With statement (4), the route planner generates a "straw plan" (step C), and then the system offers its own criticism (step D). The display at this point is shown in Figure 5. This self-criticism might seem strange at first glance, but consider that the system has not been informed of any evaluation criteria for the goals, so does not know whether the time for the trip is a critical factor. In (5), the manager indicates that the delay is not a problem by accepting the straw plan, and then moves on to a new focus (implicitly) by adding constraints relevant to the subgoal of getting to Syracuse (steps A and B). The system does not have enough constraints to create a straw plan solution, so (6) prompts for additional constraints. Statement (7) specifies these additional constraints (step B), and statement (8) proposes a straw plan route (step C) and potential criticisms (step D). In this case, the manager adds additional constraints in (9) that address the problems the system mentioned and a new solution is generated as part of interaction (10). The rest of the dialogue continues in much

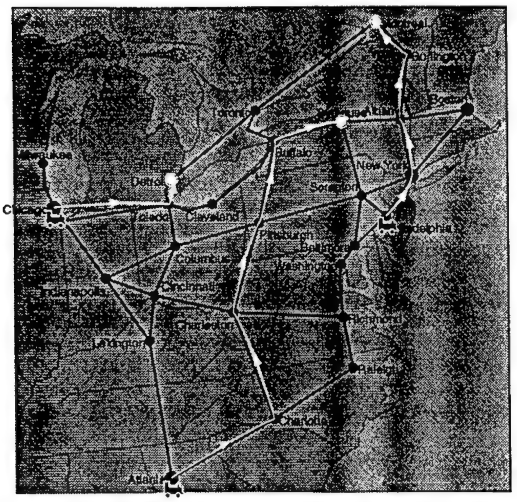


Figure 6: Final TRAINS-95 System Map Display

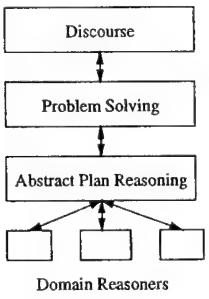


Figure 7: Mixed-initiative planning architecture

the same way and doesn't raise any substantially new issues. The final display is shown in Figure 6.

To support this model of mixed-initiative planning, we are developing a four layer architecture that generalizes the TRAINS-95 system architecture. This is shown in Figure 7. The *discourse level* maintains information important for reference identification and for speech act interpretation and generation. It maintains what objects have been recently mentioned as well as the immediate discourse situation including what speech act was performed last and what discourse obligations each agent has. While critical to the overall system, the other three levels are more centrally concerned with the planning process.

The *problem solving level* maintains meta-level information about the problem solving tasks, similar to the problem solving level actions described in [Litman and Allen, 1987] and [Lambert and Carberry, 1991]. Actions at this level are problem solving actions such as

"solve goal directly," "decompose goal into subproblems," "resolve resource conflict," and so on. Note that we need explicit representations of such actions because they can be discussed in the dialogue. Either agent might request the other to do one of these actions, or discuss how one might do the action, or assign general responsibility to an agent to perform this type of action throughout the interaction. This level supports processes such as identifying the task desired (*e.g.*, distinguishing between an elaboration of the plan for the current goal and shifting to another goal). It does this by maintaining an abstract tree of goals and information on which part of the problem is currently in focus. It also supports the process of determining the scope of problem solving actions such as cancelations and modifications. By maintaining a history of previous problem solving states, it supports intelligent "undo" commands and the explicit discussion of the history of the problem solving interaction. Finally, it supports discussion of the problem solving strategy to be used, and can maintain ordering constraints on when certain problem solving tasks should be performed.

The problem solving level reasons at the meta-level about the plans under consideration. These plans are themselves represented in terms of an *abstract plan representation level.* This representation captures aspects of plans independent of the details of reasoning about them. It can perform operations such as plan recognition in cases where deep domain reasoning is unnecessary, *e.g.*, identifying which plan achieves a particular goal. It provides a uniform representation of plans suitable for use by other components of the system, such as the problem solving, discourse, and generation components.

When more sophisticated domain-level reasoning is required, we rely on the services of a set of *domain reasoners.* The abstract plan representation level manages the interface between the mixed-initiative system and these various domain specialists, matching open issues with reasoners and coordinating the responses. These reasoners might be, for example, a scheduler or an Internet agent that can retrieve information. They are likely to include traditional planning technologies that are invoked in a context that is sufficiently well-specified for them to function effectively. We call these components "specialists" to emphasize that they need to be quick and effective in a well-defined domain, although they definitely need not always produce "perfect" answers.

The key to the integration of such components is the ability to specify and reason about their capabilities. That is, the abstract plan reasoner needs to be able to reason about which specialists might be suitable for what tasks, and interpret their results. This is complicated by the desire to use existing components "off the shelf" as much as possible. We are currently looking at using KQML [Finin *et al.*, 1994] for part of this purpose, as well as other work on specifying agent capabilities [Tate, 1995].

4 Current and Future Directions

Our experiences with TRAINS-95 have opened up many research directions that we are currently pursuing, and this paper has already discussed several of them. In this section we provide a brief survey of some additional issues for current and future work.

First, although we have argued strongly for an interactive, language-based approach to plan reasoning, this is not in itself enough. In fact, our strategy of attempting to instantiate the plan and then discuss and correct it will need to be generalized as the plans become larger and more detailed. This problem of plan presentation is pervasive, and requires work on many aspects of multi-modal interaction, since clearly drawing a huge graph with Lisp-like labels is not going to be sufficient. We believe that the dialogue-based approach provides the context in which presentation decisions can be made effectively. The TRAINS-95 system provides the infrastructure for examining the multi-modal aspects of such communication.

Second, and something of a corollary to this, is the problem of the semantics of external displays or data sources. That is, many systems are currently being proposed to make use of existing resources, such as weather servers or radar displays. However, while it's one thing to fetch the latest weather map via the Internet and display it, it's quite another to be able to discuss what is being displayed intelligently. We have emphasized how context is the big benefit of dialogue-base systems—the other side of the coin is that to make use of something in a dialogue requires that it fit into the context. Part of the solution to this lies in the type of capability descriptions that we discussed in the previous section. We are also moving towards use of a uniform display language, via a translator if necessary, in order to be able to understand exactly what is being displayed where. The impact of this on the ability to "plug and play" components has yet to be seen.

Third, concentrating on the type of planning that goes on in mixed-initiative systems, it is clear that much of it can be seen as replanning, in the sense of debugging a plan ala Sussman [Sussman, 1974]. We are looking at the various approaches to replanning (*e.g.*., [Kambhampati, 1989; Kambhampati and Hendler, 1992]), but to some extent these seem to be burdened by the same constraints that make traditional planning systems unsuitable for direct use in the mixed-initiative situation.

Fourth, we have been concerned from the outset with the evaluation of our work, that is, how to know if we are making progress. This has traditionally been a problem for interactive systems, and to some extent for planners. Our current system has built into it a simple set of task-related metrics that are recorded for each interaction. As we refine those metrics, we can explore whether particular strategies are better than others at getting the task done, or whether the presence of certain components helps or hinders performance.

In recent work [Allen *et al.*, 1996], we have evaluated the effects of input mode (speech vs. keyboard) on task performance and explored user input-mode preferences. Task performance was evaluated in terms of the amount of time taken to arrive at a solution and the quality of the solution was determined by the amount of time needed to travel the planned routes. Sixteen subjects for the experiment were recruited from undergraduate computer science courses. None of them had previous experience with the system, and in fact only five reported that they had previously used a speech recognition system. A preliminary analysis of the experimental results shows that the plans generated when speech input was used were of similar quality to those generated when keyboard input was used. However, the time needed to develop the plan was significantly lower when speech input was used. Averaging

over five scenarios, problems were solved using speech in 68% of the time using keyboard, despite speech recognition rates averaging only about 75%! Although still rough, we take these results to support our approach to building robust interactive systems by treating the interaction as a dialogue.

The point as far as mixed-initiative planning is concerned is that with only a two-minute training video, and despite low speech recognition rates and the prototypical status of the system, 73 of the 80 sessions resulted in successful completion of the route planning task. We suspect that this is probably a record for naive users of AI planning systems. Of course, the task was very simple, and it remains to be seen whether we can maintain this level of effectiveness as we move into a more complex planning domain.

Finally, we are continually improving the TRAINS system. The next version will include many of the ideas presented in these last sections of the paper, as well as a richer transportation domain involving cargoes and different vehicle types. This will increase the complexity of the planning task, as we noted above was necessary for further evaluation. Several parts of the TRAINS-96 system have already been re-implemented, including new infrastructure components, KQML message-passing, and the separation of many of the functions of the "discourse manager" into separate modules. Work is currently proceeding on separating problem solver and domain reasoner functionality, and increasing the complexity of the domain. Our goal is to always have the most recent version of the system running and available for demos, and we hope to have a version of it running over the World-Wide Web.

5 Conclusion

The TRAINS-95 system is a concrete first step towards a mixed-initiative planning assistant. We have demonstrated the feasibility of the dialogue-based approach to interactive planning, and have developed a substantial infrastructure for future research.

The TRAINS-95 system is being developed as part of the TRAINS Project [Allen *et al.*, 1995], a long-term research effort to develop intelligent assistants that interact with their human managers using natural language. More information is available via the World-Wide Web at URL: `http://www.cs.rochester.edu/research/trains/`.

6 Acknowledgements

This material is based upon work supported by ARPA – Rome Laboratory under research contract no. F30602-95-1-0025, the U.S. Air Force – Rome Laboratory under research contract no. F30602-91-C-0010, and by the Office of Naval Research under research grant no. N00014-95-1-1088. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

7 References

[Allen, 1994] James F. Allen, *Natural Language Understanding*, Benjamin Cummings, 2nd edition edition, 1994.

[Allen *et al.*, 1996] James F. Allen, Bradford Miller, Eric K. Ringger, and Teresa Sikorski, "Robust Understanding in a Dialogue System". Submitted for publication, January 1996.

[Allen *et al.*, 1995] James F. Allen, Lenhart K. Schubert, George Ferguson, Peter Heeman, Chung Hee Hwang, Tsuneaki Kato, Marc Light, Nathaniel G. Martin, Bradford W. Miller, Massimo Poesio, and David R. Traum, "The TRAINS Project: A case study in building a conversational planning agent," *Journal of Experimental and Theoretical AI*, 7:7–48, 1995. Also available as University of Rochester, Dept. of Computer Science TRAINS Technical Note 94-3.

[Ferguson, 1995] George Ferguson, *Knowledge Representation and Reasoning for Mixed-Initiative Planning*, PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, January 1995. Available as Technical Report 562.

[Finin *et al.*, 1994] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire, "KQML as an Agent Communication Language," in *Proc. Third International Conference on Information and Knowledge Management (CIKM-94)*. ACM Press, November 1994.

[Heeman and Allen, 1995] Peter A. Heeman and James F. Allen, "The TRAINS-93 Dialogues," TRAINS Technical Note 94-2, Dept. of Computer Science, University of Rochester, Rochester, NY, March 1995.

[Huang *et al.*, 1992] Xuedong Huang, Fileno Alleva, Hsiao-Wuen Hon, Mei-Yu Hwang, and Ronald Rosenfeld, "The SPHINX-II Speech Recognition System: An Overview," Technical Report CS-92-112, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, January 1992.

[Kambhampati, 1989] Subbarao Kambhampati, *Flexible reuse and modification in hierarchical planning: A validation structure based approach*, PhD thesis, Department of Computer Science, University of Maryland, College Park, MD, 1989. Available as CS Technical Report 2334.

[Kambhampati and Hendler, 1992] Subbarao Kambhampati and James A. Hendler, "A validation-structure-based theory of plan modification and reuse," *Artificial Intelligence*, 55:193–258, 1992.

[Lambert and Carberry, 1991] Lynn Lambert and Sandra Carberry, "A tripartite plan-based model of dialogue," in *Proc. Twenty-Ninth Annual Meeting of the Association for Computational Linguistics*, pages 47–54, Berkeley, CA, 18–21 June 1991. University of California.

[Litman and Allen, 1987] Diane Litman and James F. Allen, "A plan-recognition model for subdialogues in conversations," *Cognitive Science*, 11(2), 1987.

[Pollack, 1992] Martha E. Pollack, "The uses of plans," *Artificial Intelligence*, 57, 1992.

[Ringger, 1995] Eric K. Ringger, "A Robust Loose Coupling for Speech Recognition and Natural Language Understanding," Technical Report 592, Dept. of Computer Science, University of Rochester, Rochester, NY, September 1995.

[Sacerdoti, 1977] E.D. Sacerdoti, *A Structure for Plans and Behaviour*, Elsevier, North-Holland, New York, NY, 1977.

[Sussman, 1974] Gerald J. Sussman, "The virtuous nature of bugs," in *Proc. First Conference of the Society for the Study of AI and the Simulation of Behaviour*, Brighton, UK, 1974. Sussex University.

[Tate, 1995] Austin Tate, "The O-Plan Knowledge Source Framework," ARPA/RL O-Plan TR 21, Artifical Intelligence Applications Institute, University of Edinburgh, Edinburgh, UK, March 1995.

Appendix C

Arguing About Plans: Plan Representation and Reasoning for Mixed-Initiative Planning*

Abstract

We consider the problem of representing plans for mixed-initiative planning, where several participants cooperate to develop plans. We claim that in such an environment, a crucial task is *plan communication*: the ability to suggest aspects of a plan, accept such suggestions from other agents, criticize plans, revise them, *etc.*, in addition to building plans. The complexity of this interaction imposes significant new requirements on the representation of plans. We describe a formal model of plans based on defeasible argument systems that allows us to perform these types of reasoning. The arguments that are produced are explicit objects that can be used to provide a semantics for statements about plans.

1 Introduction

Mixed-initiative planning involves the cooperation of two or more agents collaboratively constructing and possibly executing a plan. This includes situations ranging from automated planning assistants, where a machine aids a human in constructing a plan, to situations with multiple autonomous agents attempting to co-ordinate their activities towards a common goal. In all these situations, the agents need to talk about plans. This paper develops a representation that supports such plan communication. While we have used this to support a natural language mode of communication, it applies equally well to more formally defined

*This material is based on: George Ferguson and James F. Allen, "Arguing About Plans: Plan Representation and Reasoning for Mixed-Initiative Planning," *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, pp. 43–48, Chicago, IL, 13–15 June, 1994.

communication languages such as two machines might use, or as might be used as the back-end for a graphical human-machine interface.

The traditional representations of plans are inadequate for supporting communication about plans. Formal models of plans (*e.g.*, [McCarthy and Hayes, 1969],[Chapman, 1987]) typically define a plan as a sequence of actions. If agents communicated plans by simply listing the sequence of actions to be performed, then this might be adequate. But this is not what happens. To see how agents communicate plans, we designed an experiment that involves two people collaborating on forming plans in a transportation planning domain involving train scheduling. The subjects could not see each other, and could only communicate by speaking into a microphone. Over thirteen hours of interaction have been collected and transcribed (see [Gross *et al.*, 1993]). In no case did one agent simply describe the plan by describing a sequence of actions. Rather, agents identified the overall goal, identified subgoals to focus on, identified important actions in the plan, stated relevant facts that would help in the development of the plan, identified problems with what the other agent proposed, confirmed what the other agent suggested, and requested clarification when suggestions were not fully understood. Many of the details of the plan are never mentioned at all, and yet are implicitly agreed upon by the agents.

Clearly, the representation of a plan as a sequence of actions accounts for very little of the actual interactions that occur in a mixed-initiative planning scenario. This is not to say that we must reject the notion of a plan as a sequence of actions. It only points out that a planning system that engages in mixed-initiative planning must use a much richer representation. Similar conclusions have been made by researchers interested in plan execution monitoring and plan modification and reuse [Hayes, 1975; Drummond, 1989; Kambhampati and Hendler, 1992; Kambhampati, 1992]. For these applications, it is important to represent the reasons why an action is being performed, so that an appropriate response can be made if the action fails. It is also important to record which effects of an action are important for the plan in order to verify whether an action succeeded or failed in the first place. This same information is crucial in plan communication. It allows one agent to understand the other agent's motivation behind a suggested course of action, which helps in identifying the specific role that the suggested action should play in the plan.

Consider an example. Two agents, A and B, are cooperating to construct a plan to transport medical supplies to some location. To get the supplies there, the agents need to first move them overland to the port, and then carry them by ship. A cargo ships leaves every day at between 4:00 and 6:00. If the supplies are moved by train to the ship, they will arrive at 5:00. If they are moved by truck, they will arrive at 3:00. Moving them by truck, however, will be three times as expensive. Given this scenario, there are competing constraints on a solution. Ideally, one would like a plan that guarantees to get the supplies there as soon as possible, while spending the least amount of money. Such a solution is not possible here, however. Furthermore, there is no single, global evaluation measure on the worth of plans. Each of the agents may have different priorities in evaluating plans, possibly focusing on one factor and ignoring the others.

Given this setting, one possible interaction between the two agents is as follows:

1. Agent A suggests shipping the supplies by train;
2. Agent B points out that the ship might leave at 4:00, and thus the supplies would miss today's ship;
3. Agent A points out that the ship might not leave until 6:00, and thus the supplies would make today's ship.

At this point, the agents are at a standoff. They could continue to argue in many ways. A might argue that getting the supplies there today isn't important, or B might argue that the risk of missing the ship is unacceptable. Or one of them might propose another plan:

4. Agent B suggests moving the supplies by truck instead.

Assuming that agent A finds the cost of this option acceptable, there would appear to be no arguments against this plan, and thus it would be accepted by both agents.

There are many problems to address in order to support the type of interactions in the example. In this paper we are concerned with the formal representational underpinnings. The representation proposed involves applying techniques for representing arguments to a logic of time, events and action, which supports reasoning about attempting actions and their effects. We have explored the logic of time and action in depth elsewhere [Allen and Ferguson, 1994] and have applied it to plan reasoning [Allen, 1991]. Here we explore the use of explicit argument structures in a direct inference system, and show how, with a suitable logic of time and action, they provide a very rich representation both for talking about plans and for doing plan reasoning itself. This representation is being used to support plan reasoning in the TRAINS system at Rochester, where it supports plan construction and plan recognition algorithms that are used by a planning system that cooperatively builds plans with a person.

The rest of this paper is organized as follows. First we define argument systems formally, then apply the model to representing planning knowledge. The preceding example is used to illustrate the approach. Finally we discuss some of the issues raised by this work.

2 Argument Systems

In this section we present a formal description of an argument system based on those of Loui [Loui, 1987] and of Pollock [Pollock, 1987; Pollock, 1992].

2.1 Basic Definitions

We assume a logical language with an entailment relation $\models$, and allow a set KB of propositions that specify domain constraints against which arguments can be evaluated.

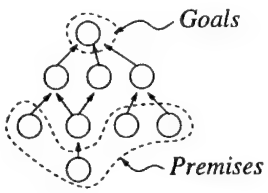


Figure 1: Graphical representation of an argument

Definition 1 *An* ARGUMENT STEP *is a pair* $\langle \Phi, p \rangle$*, written "*$\Phi \rightarrow p$*," where* Φ *is a set of propositions (the* PREMISES*) and* p *is a single proposition (the* CONCLUSION*).*

An argument step is to be read something like "Φ is reason to believe p," or "The truth of Φ lends support to the truth of p."

Am agent's knowledge base includes a set of argument steps Δ that the agent uses in its reasoning. Note that in the case where we have large numbers of potential arguments, Δ may be specified schematically.

Definition 2 *An* ARGUMENT *is a set of argument steps* $\{\langle \Phi_i, p_i \rangle\}$ *drawn from* Δ*.*

Definition 3 *The* PREMISES *of an argument* A *are those propositions in the conclusions of steps with empty premises,* i.e., $\{p | \langle \emptyset, p \rangle \in A\}$*. The* CONCLUSIONS *of an argument* A *are the union of the premises of steps with empty conclusions,* i.e., $\bigcup\{\Phi | \langle \Phi, \epsilon \rangle \in A\}$*.*

We require that arguments be non-circular and that both $Premises(A)$ and $Goals(A)$ be non-empty for any argument A. In this case, arguments correspond to directed, acyclic graphs labeled with propositions, such as illustrated in Figure 1. An argument step $\langle \Phi, p \rangle$ corresponds to a node labeled by p with children each labeled by an element of Φ. The sources of the graph correspond to $Premises(A)$, the sinks to $Goals(A)$.

2.2 Conflict and Defeat

What makes argument systems interesting is the way arguments can conflict over the status of some proposition. In this case, we need to define how certain arguments are to be preferred over others.

Definition 4 *Two argument steps* $\langle \Phi, p \rangle$ *and* $\langle \Psi, q \rangle$ CONFLICT *if* p *and* q *are inconsistent.*

In Pollock's system, this form of conflict is called *rebuttal*, and a second form, called *undercutting*, is also permitted. The intuition is that in rebutting conflict the value of some proposition is at issue, while in undercutting conflict it is the applicability of a defeasible rule that is in doubt. In our approach, following Loui, we will impose certain criteria of

defeat (below) which to some extent obviate the need for undercutting. We leave open the possibility, however, that it may be required in the future.

Given two arguments that conflict, we are interested in preferring certain arguments over others. The following definition, adapted from Loui, makes this explicit:

Definition 5 *An argument A is* AT LEAST AS SPECIFIC *as an argument B if there is a step $\langle \Phi, p \rangle \in A$ and a step $\langle \Psi, q \rangle \in B$ such that the steps conflict and $\Phi \models \Psi$. An argument is* MORE SPECIFIC *than another argument if it is at least as specific and the converse is not the case.*

Finally, we can put these pieces together and propose the following:

Definition 6 *An argument A* DEFEATS *an argument B if A conflicts with B and A is at least as specific as B.*

Definition 7 *An argument A is* UNDEFEATED *if there is no argument B that conflicts with A that is not itself defeated.*

There are complications in applying this somewhat circular definition to determine which arguments are undefeated. Pollock [Pollock, 1992], for example, considers such phenomena as *collective defeat* and *self-defeat.* The details are not necessary to an appreciation of the formalism as regards planning.

Also, there might be other grounds for preferring one argument over another besides specificity. Loui [Loui, 1987] considers *evidence*, *directness*, and *preferred premise*, for example. As well, there might be domain-specific criteria, such as resource use or time limits in a planning context. The specificity principle is generally accepted as appropriate and is sufficient for what follows.

3 Plan Reasoning Using Arguments

In this section we describe the application of argument systems to planning problems. We begin by describing the representation of causal knowledge using defeasible rules and consider the qualification problem in this light. We then present a formalization of the mixed-initiative planning example presented earlier. This is a short but interestingly complex example of a planning problem that highlights the incrementality of argumentation and the use of specificity. The example includes reasoning about uncertainty and external events.

The approach is to express our defeasible knowledge about actions and their preconditions and effects using rules in Δ. Arguments can then be constructed supporting the fact that executing certain actions will achieve certain goals. These arguments may be in conflict with or defeated by other arguments. Undefeated arguments represent plans that will succeed,

given what we know and how long we searched for counter-arguments. Of course, plans can be compared or evaluated according to other factors besides defeat, such as resource usage or time constraints. Note that, among other things, this approach distinguishes goals from other effects and makes the assumptions underlying the plan explicit and thus available for reasoning.

3.1 Knowledge Representation

To keep the presentation as concise as possible, we will use a very simple representation of time, namely a discrete model where, given a time t, $n(t)$ is the next time. The representation can easily be generalized to either a situation calculus representation or the more expressive interval temporal logic [Allen and Ferguson, 1994] that we actually use. An important point is that action attempts must be distinguished from their effects.

In order to do this, we introduce *events* as cognitive objects that correspond to "something that happened." Event predicates are used to organize our causal knowledge about the domain and are organized in a type hierarchy. For example, a predicate like $Load(e_t)$ is true if the event e_t describes a loading event that occurred at time t. Rules describe the necessary conditions given that an event of a particular type occurred.

In order to reason about the agent's abilities, we have to introduce *actions*. Unfortunately, actions and events are often confused, since, intuitively, for any action there is an event corresponding to that action being executed. To avoid this confusion, we will think of actions as sensory-motor *programs* that can be run by the agent; this is basically the concept of action used in the planning literature. There is a predicate $Try(\pi, t, e_t)$ which is true if program π is executed at time t (or over interval t in a more expressive logic) causing event e_t to occur. Rules specify sufficient conditions for a program's execution to result in an event of a certain type.

3.2 Causal Knowledge and Qualifications

The fundamental issue in representing knowledge for planning is how to capture the idea that execution of an action causes a certain effect, at least under certain conditions. The traditional planning view would be characterized something like the following:

$$Holds(preconds(a), t) \wedge Try(a, t, e_t) \supset Holds(effects(a), n(t)).$$

That is, if the preconditions of an action a hold at time t and we attempt action a, then the effects will hold at the next time point.[1] There are many problems with this definition, such as how to handle simultaneous actions, external events, and actions with duration. These

[1]Traditional models wouldn't have events, but we include the e_t term for comparison with the later development.

problems can be resolved, however, by using a richer temporal model, but for this paper such extensions are not important. The important idea is the intuition that actions do have preconditions, even if these are context-dependent and subject to revision.

In the context of reasoning about actual situations, what is troubling is the strength of the implication. It does not make sense with this definition, for example, to reason about execution of the action when only some of the preconditions are known to be true. Indeed, in the STRIPS model of planning, an operator cannot even be applied (*i.e.*, an action cannot be attempted) if all the preconditions do not hold. In a more philosophical sense, these axioms run into the *qualification problem*: the problem that no matter how many preconditions we write down, there will always be other things that might happen that would invalidate the axiom. If these axioms are based on the material conditional, the price of encountering an unconsidered qualification is inconsistency.

The obvious move then is to weaken the definition to use the defeasible connective "$\rightarrow$" rather than the material conditional. Then these definitions can be used in arguments that execution of an action has some effect, without saying finally that it must. We obtain something like the following, using the notation introduced in the previous section:

$$\{Holds(preconds(a), t), Try(a, t, e_t)\} \rightarrow Event(e_t)$$

where *Event* is an event-type predicate. The definition of the event can continue to use the material conditional:

$$Event(e_t) \supset Holds(effects(e_t), n(t))$$

This is not quite enough, however, because it doesn't capture the notion that somehow the precondition *should* be true before attempting the action if it is to succeed, even if we don't know for sure that it this is the case. Clearly we shouldn't state that falsity of a precondition implies that the action won't succeed, since there might be other conditions (known or unknown) under which it would. But we can say that defeasibly, as in

$$\{\neg Holds(preconds(a), t), Try(a, t, e_t)\} \rightarrow \neg Event(e_t)$$

In fact, these two rules form the extremes of a continuum of rules relating knowledge of preconditions to successful execution of actions. Suppose we know that ϕ_1 and ϕ_2 are preconditions for action a. Then a plan that takes account of only ϕ_1 is at least reasonable, if not complete or even possible given other knowledge about the world. But it is important to be able to represent such partial or incorrect plans if we are interested in doing more than simply generating simple plans. In this case, we will need additional rules in Δ corresponding to these "partial" argument steps. In fact, for every action a with (conjunctive) preconditions $\phi_1, \phi_2, \ldots, \phi_n$ and effects ψ, there corresponds a set of defeasible rules organized in a lattice of specificity. The most specific rule is the one in which all preconditions are present. The least specific rule mentions no preconditions, *i.e.*, $Try(a, t, e) \rightarrow Event(e)$, indicating that an action can always be attempted, even if an argument that doesn't take note of preconditions may not be a very strong one. There is a similar lattice of rules regarding the falsity of preconditions and the non-occurrence of events.

Some comments are in order regarding this formulation of causal rules. First, it is independent of the underlying representation of time and action, so long as action attempts can be distinguished from their effects (*i.e.*, the *Try* predicate, above). Second, the approach is different from using a modal logic to express the possibility and necessity intuitions involved in reasoning about preconditions. The most modal logic could do is allow us to write that if a precondition is true then an action is possible, and if it is false then the action necessarily fails, for example. But in the defeasible reasoning approach, degrees of support are afforded through the use of defeasible rules with more or less specific antecedents. This is much closer to our intuitions about just what a precondition is, and also closer to the way people talk about preconditions in collaborative discourse about plans.

Finally, we believe that this approach to reasoning about preconditions and effects of actions is the best way to deal with the qualification problem. That is, the agent considers qualifications as long as it can or until it has considered all that it knows about, as reflected in its set of defeasible rules. If subsequent qualifications are encountered, they can be reasoned about given more time, without denying that the plan developed previously supported its conclusions, however tenuously.

3.3 Planning Example

Recall from the example presented in the Introduction that we have to get supplies X into ship S before it leaves port P, and that there are two ways of transporting the supplies. We will denote these by two actions (programs): *sendByTrain* and *sendByTruck*. Then assuming these actions have no preconditions, their definitions are given by the following rules:

$$\{Try(sendByTrain(x,l),t,e_t)\} \rightarrow Transport(e_t,x,l,5)$$
$$\{Try(sendByTruck(x,l),t,e_t)\} \rightarrow Transport(e_t,x,l,3).$$

Time units are measured in hours. The event predicate *Transport* is defined as

$$Transport(e_t,x,l,n) \supset At(x,l,t+n),$$

The action of loading ship s with supplies x has as preconditions that both the ship and the supplies must be at the dock, yielding the following definition:

$$\{At(x,l,t), At(s,l,t), Try(load(x,s,l),t,e_t)\} \rightarrow Load(e_t,x,s,l).$$

The definition of *Load* is

$$Load(e_t,x,s,l) \supset In(x,s,t+1).$$

Finally, we have the information that the ship S leaves port P between 4:00 and 6:00. Letting T_d denote the time of departure, we have that $4 \leq T_d \leq 6$ and defeasible rules:

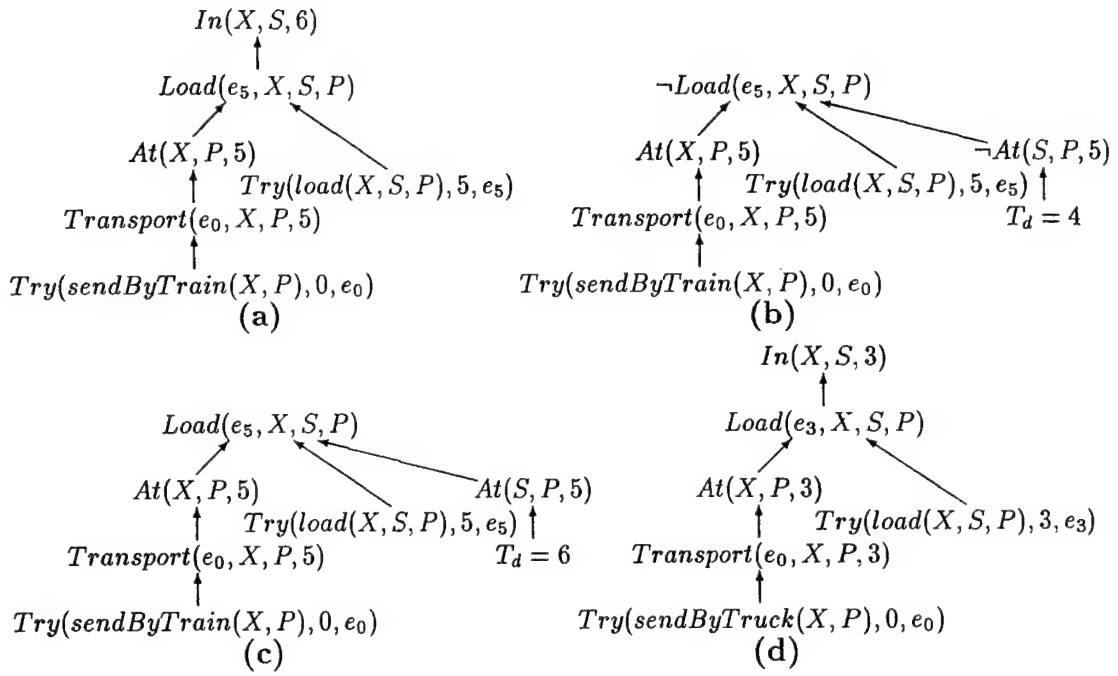


Figure 2: Four arguments about shipping supplies

$$\begin{aligned} T_d > t &\rightarrow At(S, P, t) \\ T_d \leq t &\rightarrow \neg At(S, P, t). \end{aligned}$$

Agent A's initial plan, namely shipping the the supplies by truck, is shown in Figure 2(a). Obviously, this ignores the precondition that the ship be at the dock at the time of loading. Figure 2(b) shows agent B's a *more specific* argument that argues that the loading will fail if the ship leaves at 4:00. Figure 2(c) shows the equally specific argument supporting the successful loading of the supplies if the ship hasn't left yet. Since neither argument is more specific, neither emerges as undefeated and so, intuitively, neither plan is guaranteed. This is as it should be given the uncertainty in the scenario. Figure 2(d) shows the undefeated argument for using the truck rather than the train.

The point of this example is to demonstrate the incremental development of a plan when several agents are involved. By representing the interaction as a process of argumentation, we can account for interactions that point out of difficulties with a current proposal, or that propose alternative courses of action, as well as many other interactions beyond the scope of a traditional planning system. Further, the arguments that are built up explicitly include the types of information typically added to planners to support, *e.g.*, plan modification, while being formally well-motivated in terms of defeasible reasoning. Of course, the particular example used here is not particularly difficult for classical planning systems, although it does contain some complications involving uncertainty and external events that would cause problems for many frameworks.

4 Towards A Language for Plans

The description of plans as arguments allows us to define a language in which plans are objects in the ontology. Terms denoting plans are then given a semantics in terms of plan graphs, where different predicates impose different constraints on the interpretation (*i.e.*, on the graph). There are *structural* predicates, such as *ActionIn*, *Enables*, *Premise*, *Goal*, *etc.*, that are defined in terms of the structure of the plan graph. Then there are *evaluation* predicates. These include *absolute* predicates such as *Plausible* or *Impossible* and *relative* predicates that allow plans to be compared, for example, according to resource usage or time constraints. The view of mixed-initiative planning that emerges is one where agents post constraints on the shared plan using these predicates.

This language is being developed as part of the TRAINS project [Allen and Schubert, 1991; Ferguson, 1994], an effort to construct an intelligent planning assistant that is conversationally proficient in natural language. The plan description predicates are used in the interface between the language and discourse modules and the domain plan reasoner. The manager's utterances are result in queries being made of or constraints being posted on the plan. The plan reasoner communicates the results in terms of plan description predicates that satisfy the query or that needed to be added in order to connect an utterance to the plan. These are then used to generate responses or helpful suggestions, since often information beyond the literal content of an utterance must be added in order to incorporate it.

5 Discussion

There are many questions raised by this work, but space permits discussion of only a few. First, this work has clear similarities to the work on plan modification and replanning mentioned at the outset. The difference is that rather than developing data structures for traditional planners that retain information required for other tasks, we consider planning within the context of a general defeasible reasoning system. Thus, our work can be seen as a generalization and formalization of that work which can be applied to other problems involved in plan communication. Our approach also abstracts away from the underlying representation of action (*i.e.*, STRIPS).

Konolige [Konolige, 1988] applies defeasible reasoning to reasoning about events, in particular to the Yale Shooting problem. The emphasis is on what types of defeasible arguments are important in reasoning about events as well as what information is necessary for adjudicating between these arguments. While the particular rules presented are largely an artifact of the simple representation of action, he argues convincingly for the relative merits of argumentation systems compared to indirect systems such as circumscription. He notes the ability to include knowledge about applicability of defeasible rules directly within the framework and the incremental nature of argumentation that we have also noted.

Konolige and Pollack [Konolige and Pollack, 1989] directly apply the defeasible reasoning paradigm to plan recognition in the context of plans-as-intentions. The aim is to produce plausible arguments regarding the intentions of other agents from observations of their actions, and from the undefeated arguments to extract intentions that can be ascribed to those agents. The difference between our approaches is that we see the arguments as objects to be reasoned about rather than using argumentation to reason defeasibly about intentions. These are not irreconcilable viewpoints.

6 Conclusions

We have presented an explicit representation of plans as arguments that a certain course of action under certain explicit conditions will achieves certain explicit goals. We believe that in realistic mixed-initiative planning scenarios such a representation is necessary to capture the wide range of reasoning that agents do with plans. Basing the representation on argument systems provides a formal basis both for describing these forms of reasoning and for defining the semantics of plans.

7 References

[Allen, 1991] James F. Allen, "Temporal Reasoning and Planning," in *Reasoning about Plans*, pages 1–68. Morgan Kaufmann, San Mateo, CA, 1991.

[Allen and Ferguson, 1994] James F. Allen and George Ferguson, "Actions and events in interval temporal logic," *Journal of Logic and Computation*, 4(5):531–579, 1994. A much earlier version appeared in Working notes of the Second Symposium on Logical Formalizations of Commonsense Reasoning, Austin, TX, 11–13 January, 1993.

[Allen and Schubert, 1991] James F. Allen and Lenhart K. Schubert, "The TRAINS Project," TRAINS Technical Note 91-1, Department of Computer Science, University of Rochester, Rochester, NY, 14627, May 1991.

[Chapman, 1987] David Chapman, "Planning for conjunctive goals," *Artificial Intelligence*, 32:333–377, 1987. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 537–558.

[Drummond, 1989] M. Drummond, "Situated control rules," in R.J. Brachman, H.J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 103–113, Toronto, Ont., 15–18 May 1989. Morgan Kaufmann.

[Ferguson, 1994] George Ferguson, "Domain Plan Reasoning in TRAINS-93," TRAINS technical note, Department of Computer Science, University of Rochester, Rochester. NY, 14627, 1994. To appear.

[Gross *et al.*, 1993] Derek Gross, James F. Allen, and David R. Traum, "The TRAINS-91 Dialogues," TRAINS Technical Note 92-1, Department of Computer Science, University of Rochester, Rochester, NY, 14627, June 1993.

[Hayes, 1975] Patrick J. Hayes, "A representation for robot plans," in *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pages 181–188, Tbilisi, Georgia, USSR, 3–8 September 1975. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 154–161.

[Kambhampati, 1992] Subbarao Kambhampati, "Characterizing multi-contributor causal structure for planning," in *Proceedings of the First International Conference on AI Planning Systems*, pages 116–125, College Park, MD, 15–17 June 1992. Morgan Kaufmann.

[Kambhampati and Hendler, 1992] Subbarao Kambhampati and James A. Hendler, "A validation-structure-based theory of plan modification and reuse," *Artificial Intelligence*, 55:193–258, 1992.

[Konolige, 1988] Kurt Konolige, "Defeasible argumentation in reasoning about events," in Zbigniew W. Ras and Lorenza Saitta, editors, *Methodologies for Intelligent Systems 3, Proceedings of the Third International Symposium*, pages 380–390, Turin, Italy, 12–15 October 1988. North-Holland.

[Konolige and Pollack, 1989] Kurt Konolige and Martha E. Pollack, "Ascribing plans to agents, preliminary report," in Natesa Sastri Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 924–930, Detroit, MI, 20–25 August 1989.

[Loui, 1987] R.P. Loui, "Defeat among arguments: A system of defeasible inference," *Computational Intelligence*, 3(2):100–106, 1987.

[McCarthy and Hayes, 1969] John McCarthy and Patrick J. Hayes, "Some philosophical problems from the standpoint of artificial intelligence," in B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. American Elsevier Publishing Co., Inc., 1969. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 393–435.

[Pollock, 1987] John L. Pollock, "Defeasible reasoning," *Cognitive Science*, 11:481–518, 1987.

[Pollock, 1992] John L. Pollock, "How to reason defeasibly," *Artificial Intelligence*, 57:1–42, 1992.

Appendix D

Actions and Events in Interval Temporal Logic*

Abstract

We present a representation of events and action based on interval temporal logic that is significantly more expressive and more natural than most previous AI approaches. The representation is motivated by work in natural language semantics and discourse, temporal logic, and AI planning and plan recognition. The formal basis of the representation is presented in detail, from the axiomatization of time periods to the relationship between actions and events and their effects. The power of the representation is illustrated by applying it to the axiomatization and solution of several standard problems from the AI literature on action and change. An approach to the frame problem based on explanation closure is shown to be both powerful and natural when combined with our representational framework. We also discuss features of the logic that are beyond the scope of many traditional representations, and describe our approach to difficult problems such as external events and simultaneous actions.

1 Introduction

Representing and reasoning about the dynamic aspects of the world—primarily about actions and events—is a problem of interest to many different disciplines. In AI, we are interested in such problems for a number of reasons, in particular to model the reasoning of intelligent agents as they plan to act in the world, and to reason about causal effects in the world. More specifically, a general representation of actions and events has to support the following somewhat overlapping tasks:

*This is a shortened version of a paper that appeared in *Journal of Logic and Computation*, 4(5):531–579, 1994.

1. Prediction: Given a description of a scenario, including actions and events, what will (or is most likely to) happen?

2. Planning: Given an initial description of the world and a desired goal, find a course of action that will (or is most likely to) achieve that goal.

3. Explanation: Given a set of observations about the world, find the best explanation of the data. When the observations are another agent's actions and the explanation desired is the agent's plan, and the problem is called plan recognition.

Our claim in this paper is that in order to adequately represent actions and events, one needs an explicit temporal logic, and that approaches with weaker temporal models, such as state spaces (*e.g.*, STRIPS-based approaches) and the situation calculus, either cannot handle the problems or require such dramatic extensions that one in effect has grafted an explicit temporal logic onto the earlier formalism. Furthermore, if one of these formalisms is extended in this way, the temporal logic part will dominate and the original formalism plays little role in the solution. We will primarily defend this position by proposing a specific temporal representation and showing that it can handle a wide range of situations that are often problematic for other formalisms. In particular, here are some of properties of actions and events that we feel are essential to any general representation:

1. Actions and events take time. During this time, they may have a rich structure. For instance, the event of driving my car to work involves a wide range of different actions, states of the world and other complications, yet the activity over that stretch of time is appropriately described as a single event.

2. The relationship between actions and events and their effects is complex. Some effects become true at the end of the event and remain true for some time after the event. For example, when I put a book on the table, this has the effect that the book is on the table for at least a short time after the action is completed. Other effects only hold while the event is in progress—for example holding an elevator open by pressing the "open door" button. Other effects might start after the beginning of the event and end before it does, such as my being on a bridge while driving to work. This is an effect of the action even though it is not true at the end of it. Finally, it may be the case that the effects of actions are wholly independent of the action once the action is performed, as in a rock rolling down a hill after I nudge it out of place.

3. External changes in the world may occur no matter what actions an agent plans to do, and may interact with the planned actions. Possible external events should be an important factor when reasoning about what effects an action might have. Certain goals can only be accomplished by depending on external events, whether they are a result of natural forces (such as the wind blowing enabling sailing) or the actions of other agents (such as an engine arriving with some needed cargo).

4. Actions and events may interact in complex ways when they overlap or occur simultaneously. In some cases, they will interfere with certain effects that would arise if the events were done in isolation. In other cases the effects may be additive. And in still other cases, the effect of performing the two actions may be completely different from the effects of each in isolation.

5. Knowledge of the world is necessarily incomplete and unpredictable in detail, thus reasoning about actions and events can only be done on the basis of certain assumptions. No plan is foolproof, and it is important that a formalism makes the necessary assumptions explicit so that they can be considered in evaluating plans.

Our aim is to develop a general representation of actions and events that supports a wide range of reasoning tasks, including planning, explanation, and prediction, but also natural language understanding, and commonsense reasoning in general. Most previous work tends to address only a subset of these problems. The STRIPS-based planners (*e.g.*, TWEAK [13], SIPE [63], SNLP [39]), for instance, only address the planning problem, while work in the situation calculus (*e.g.*, [41; 11]) has primarily focussed on the prediction problem or on using it as an abstract theory for planning (*e.g.*, [26]). Natural language work, on the other hand, typically only deals with commonsense entailments from statements about actions and events, sometimes with a focus on plan recognition and explanation (*e.g.*, [4; 29; 53]). Our representation is intended to serve as a uniform representation to support all these tasks. As a result, we try to avoid introducing any specific syntactic constructs that support only one reasoning task. Knowledge should be encoded in a way so that it is usable no matter what reasoning task is currently being performed.

Section 2 outlines our intuitions about actions and events, and briefly explores the two predominant models of action: the situation calculus and the state-based STRIPS-style representations. As typically formulated, neither is powerful enough for the issues described above. Section 3 then introduces Interval Temporal Logic, first defining the temporal structure, then introducing properties, events, and actions. Section 4 demonstrates how the interval logic can be used to solve a set of simple problems from the literature in order to facilitate comparison with other approaches. The key place where other approaches have difficulty is in representing external events and simultaneous actions. Section 5.1 explores the implications of external events in detail, and Section 5.3 explores interacting simultaneous actions.

2 Representing Actions and Events

Before starting the formal development, we will attempt to describe the intuitions motivating the representation. We will then consider why the most commonly accepted representations of action in AI will not meet our needs.

2.1 Intuitions about Actions and Events

The first issue concerns what an event is. We take the position that events are primarily linguistic or cognitive in nature. That is, the world does not really contain events. Rather, events are the way by which agents classify certain useful and relevant patterns of change. As such, there are very few restrictions on what an event might consist of except that it must involve at least one object over some stretch of time, or involve at least one change of state. Thus the very same circumstances in the world might be described by any number of events. For instance, consider a circumstance in which a glass fell off a table and broke on the floor. This already is one description of what actually happened. The very same set of circumstances could also be described as the event in which a glass broke, or in which someone was woken by the noise of the breaking glass. Each of these descriptions is a different way of describing the circumstances, and each is packaged as a description of an event that occurred. No one description is more correct than the other, although some may be more informative for certain circumstances, in that they help predict some required information, or suggest a way of reacting to the circumstances.

Of course, the "states" of the world referred to above are also ways of classifying the world, and are not inherent in the world itself either. Thus, the same set of circumstances described above might be partially described in terms of the ball being red. Given this, what can one say about the differences between events and states? Intuitively, one describes change and the other describes aspects that do not change. In language, we say that events occur, and that states hold. But it is easy to blur these distinctions. Thus, while the balling falling from the table to the floor clearly describes change and hence describes an event, what about the circumstance where an agent John holds the door shut for a couple of minutes. While the door remains shut during this time and thus doesn't change state, it seems that John holding the door shut is something that occured and thus is like an event. These issues have been studied extensively in work on the semantics of natural language sentences. While there are many proposals, everyone agrees on a few basic distinctions (*e.g.*, [59; 45; 16]). Of prime relevance to us here are sentences that describe *states* of the world, as in "The ball is red," or "John believes that the world is flat," and sentences that describe general ongoing *activities* such as "John ran for an hour," and *events* such as "John climbed the mountain." Each of these types of sentences has different properties, but the most important distinctions occur in their relation to time. All these sentences can be true over an interval of time. But when a state holds over an interval of time t, one can conclude that the state also held over subintervals of t. Thus, if the ball is red during the entire day, then it is also red during the morning. This property is termed *homogeneity* in the temporal logic literature. Events, on the other hand, generally have the opposite property and are anti-homogeneous: If an event occurs over an interval t, then it doesn't occur over a subinterval of t, as it would not yet be completed. Thus, if the ball dropped from the table to the floor over time t, then over a subinterval of t is would just be somewhere in the air between the table and floor. Activities, on the other hand, fall in between. They may be homogenous, as in the holding the door shut example above, but they describe some dynamic aspect of the world

like events. This type of distinction must appreciated by any general purpose knowledge representation for action and change.

Finally, a word on actions. The word "action" is used in many different senses by many different people. For us, an action refers to something that a person or robot might do. It is a way of classifying the different sorts of things than an agent can do to affect the world, thus it more resembles a sensory-motor program than an event. By performing an action, an agent causes an event to occur, which in turn may cause other desired events to also occur. For instance, I know how to perform an action of walking that I may perform in the hope of causing the event of walking to my car. Some theories refer to the event that was caused as the action, but this is not what we intend here. Rather, we will draw on an analogy with the robot situation, and view actions as programs. Thus, performing an action will be described in terms of running a program.

2.2 The Situation Calculus

The situation calculus means different things to different researchers. In its original formulation [41], which we will call the general theory of the situation calculus, situations are introduced into the ontology as a complete snapshot of the universe at some instant in time. In effect, the situation calculus is a point-based temporal logic with a branching time model. In its most common use, which we will call the constructive situation calculus, it is used in a highly restricted form first proposed by Green [26], in which the only way situations are introduced is by constructing them by action composition from an initial state. This practice has attracted the most attention precisely because the formalism is constructive—specifically, it can be used for planning. To construct a plan for a goal G, prove that there exists a situation s in which G holds. In proving this, the situation is constructed by action composition, and thus the desired sequence of actions (the plan) can be extracted from the proof. As others have pointed out (*e.g.*, [52]), most of the criticisms about the expressibility of the situation calculus concern the constructive form of it rather than the general theory. Our position is that the constructive situation calculus is a limited representation, especially in dealing with temporally complex actions and external events. The general theory, on the other hand, is much richer and can be extended to a model much closer to what we are proposing, but at the loss of its constructive aspect. This will be explored further after the Interval Temporal Logic has been introduced.

2.3 The STRIPS Representation

The STRIPS representation [20] adds additional constraints to the situation calculus model and is used by most implemented planning systems built to date. In STRIPS, a state is represented as a finite set of formulas, and the effects of an action are specified by two sets of formulas: the delete list specifies what propositions to remove from the initial state, and the add list specifies the new propositions to add. Together, they completely define the transition

```
STACK(a,b)
        preconds:  (Clear a) (Clear b)
        delete:    (Clear b)
        add:       (On a b)
```

Figure 1: Actions as state change in STRIPS

between the initial state and the resulting state. Figure 1 shows a simple Blocks World action that involves placing one block on top of another (the STACK action). The preconditions on an action indicate when the action is applicable—in this case it is applicable whenever both blocks are clear. The effects state that one block is now on top of the other (the add list) and that the bottom block is not clear (the delete list). The operation for constructing a resulting state applies the delete list first and then asserts the add list.

Like the situation calculus, STRIPS-style actions are effectively instantaneous and there is no provision for asserting what is true while an action is in execution. Also, since the state descriptions do not include information about action occurrences, such systems cannot represent the situation where one action occurs while some other event or action is occurring. Such a case would violate the STRIPS assumptions, which assume that the world only changes as the result of a single action by the agent, and that the action definitions completely characterize all change when an action is done. Of course, these assumptions would not be valid if simultaneous actions or external events were allowed.

The STRIPS assumptions appear to be fundamentally incompatible with interacting simultaneous actions and external events. They hold only in worlds with a single agent, who can only do one thing at a time, and where the world only changes as the result of the agent's actions. However, they are so ingrained in the planning literature that many researchers don't even acknowledge their limitations. In some sense, they have become part of the definition of the classical planning problem.

3 Interval Temporal Logic

Having described our motivations, we now start the development of the temporal logic. We start by describing the basic temporal structure to be used in the logic, namely the interval representation of time developed by Allen [1; 2] and discussed in detail in [6]. We then describe the temporal logic used to represent knowledge of properties, events, and actions. We conclude this section with a comparison of related formalisms. Subsequent sections will explore how the representation supports reasoning about events and actions, especially in complex situations with external events and simultaneous actions.

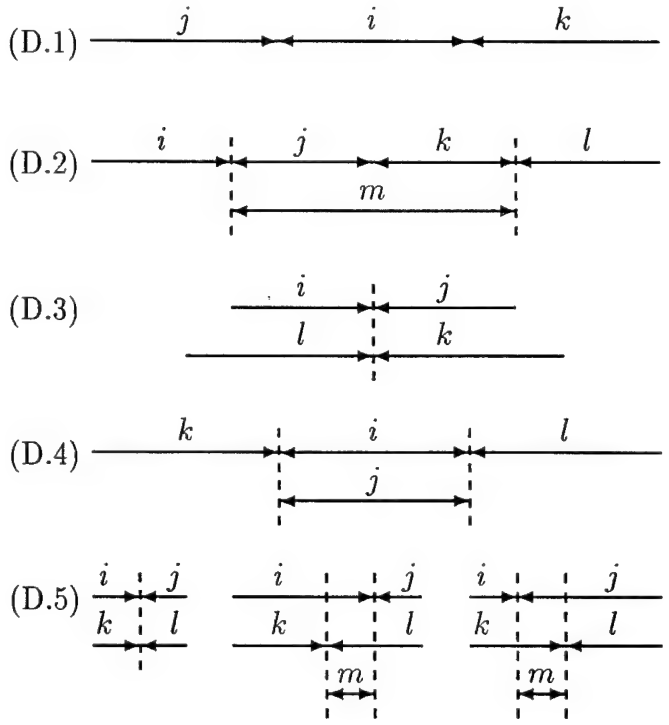


Figure 2: The axiomatization of time periods

3.1 The Structure of Time

The temporal structure we assume is a simple linear model of time. Notions of possibility that are introduced in branching time models or the situation calculus would be handled by introducing a separate modal operator to represent possibility explicitly. Since such a modal operator is needed in general anyway, there seems no need to build it into the temporal structure. The temporal theory starts with one primitive object, the time period, and one primitive relation: *Meets*.

A time period intuitively is the time associated with some event occuring or some property holding in the world. Intuitively, two periods m and n meet if and only if m precedes n, yet there is no time between m and n, and m and n do not overlap. The axiomatization of the *Meets* relation is as follows, where i, j, k, l, and m are logical variables restricted to time periods. The axioms are presented graphically in Figure 2. First, there is no beginning or ending of time and there are no semi-infinite or infinite periods. In other words, every period has a period that meets it and another that it meets:

$$\forall i \,.\, \exists j, k \,.\, Meets(j, i) \wedge Meets(i, k). \tag{D.1}$$

Second, periods can compose to produce a larger period. In particular, for any two

periods that meet, there is another period that is the "concatenation" of them. This can be axiomatized as follows:

$$\forall i,j,k,l\,.\,Meets(i,j) \wedge Meets(j,k) \wedge Meets(k,l) \supset \exists m\,.\,Meets(i,m) \wedge Meets(m,l). \tag{D.2}$$

As a convenient notation, we will often write $j + k$ to denote the interval that is the concatenation of intervals j and k. This functional notation is justified because we can prove that the result of $j + k$ is unique [6].

Next, periods uniquely define an equivalence class of periods that meet them. In particular, if i meets j and i meets k, then any period l that meets j must also meet k:

$$\forall i,j,k,l\,.\,Meets(i,j) \wedge Meets(i,k) \wedge Meets(l,j) \supset Meets(l,k). \tag{D.3}$$

These equivalence classes also uniquely define the periods. In particular, if two periods both meet the same period, and another period meets both of them, then the periods are equal:

$$\forall i,j,k,l\,.\,Meets(k,i) \wedge Meets(k,j) \wedge Meets(i,l) \wedge Meets(j,l) \supset i = j. \tag{D.4}$$

Finally, we need an ordering axiom. Intuitively, this axiom asserts that for any two pairs of periods, such that i meets j and k meets l, then either they both meet at the same "place," or the place where i meets j precedes the place where k meets l, or vice versa. In terms of the meets relation, this can be axiomatized as follows, where the symbol "$\otimes$" means "exclusive-or":

$$\begin{aligned}\forall i,j,k,l\,.\,(Meets(i,j) \wedge Meets(k,l)) \supset \\ Meets(i,l) \otimes (\exists m\,.\,Meets(k,m) \wedge Meets(m,j)) \otimes \\ (\exists m\,.\,Meets(i,m) \vee Meets(m,l)).\end{aligned} \tag{D.5}$$

Many of the properties that are intuitively desired but have not yet been mentioned are actually theorems of this axiomatization. In particular, it can be proven that no period can meet itself, and that if one period i meets another j then j cannot also meet i (*i.e.*, finite circular models of time are not possible).

With this system, one can define the complete range of the intuitive relationships that could hold between time periods. For example, one period is before another if there exists another period that spans the time between them, for instance:

$$Before(i,j) \equiv \exists m\,.\,Meets(i,m) \wedge Meets(m,j).$$

Figure 3 shows each of these relationships graphically (equality is not shown). We will use the following symbols to stand for commonly-used relations and disjunctions of relations:

$Meets(i,j)$	$i : j$		
$Before(i,j)$	$i \prec j$	$Before(i,j) \vee Meets(i,j)$	$i \prec\!: j$
$During(i,j)$	$i \sqsubset j$	$During(i,j) \vee i = j$	$i \sqsubseteq j$

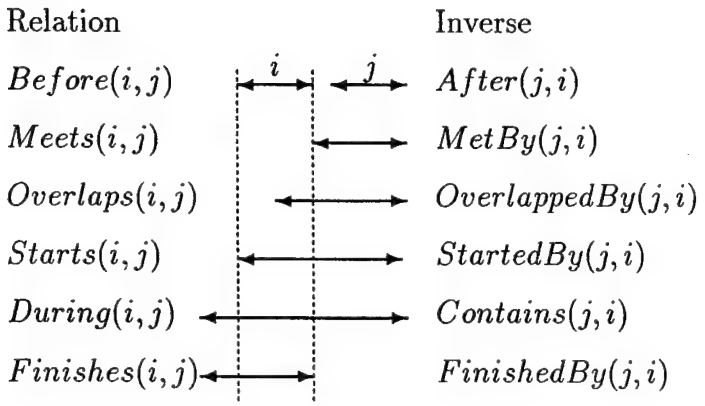


Figure 3: The possible relations between time periods (equality not shown)

Finally, an important relationship between two periods is *Disjoint*: two intervals are disjoint if they do not overlap in any way. We write this as "$i \bowtie j$" and define it by

$$i \bowtie j \equiv i \prec: j \vee j \prec: i.$$

The computational properties of the interval calculus and algorithms for maintaining networks of temporal constraints are presented in [1; 61; 62].

A period can be classified by the relationships that it can have with other periods. For example, we call a period that has no subperiods (*i.e.*, no period is contained in it or overlaps it) a *moment* and a period that has subperiods, an *interval*. In addition, we can define a notion of time point by a construction that defines the beginning and ending of periods. It is important to note that moments and points are distinct and cannot be collapsed. In particular, moments are true periods and may meet other periods, and if $Meets(i, m) \wedge Meets(m, j)$ for any moment m, then i is before j. Points, on the other hand, are not periods and cannot meet periods. Full details can be found in [6].

Any semantic model that allows these distinctions would be a possible model of time. In particular, a discrete time model can be given for this logic, where periods map to pairs of integers $\langle I, J \rangle$ where $I < J$. Moments correspond to pairs of the form $\langle I, I + 1 \rangle$, and points correspond to the integers themselves. A similar model built out of pairs of real numbers does not allow moments. A more complex model can be specified out of the reals, however, that does allow continuous time, or models that are sometimes discrete and sometimes continuous are possible. Ladkin and Maddux [33] have characterized the set of possible models as precisely the arbitrary unbounded linear orders.

3.2 Interval Temporal Logic

The most obvious way to add times into a logic is to add an extra argument to each predicate. For example, in a nontemporal logic, the predicate *Green* might denote the set of green

objects. Thus, a formula such as $Green(frog13)$ would be true only if the object named by the term, $frog13$, is in the set of green objects. To make this a temporal predicate, a time argument is added and *Green* now denotes a set of tuples consisting of all green objects with the times at which they were green. Thus, the proposition $Green(frog13, t1)$ is true only if the object named by $frog13$ was green over the time named by $t1$.

By allowing time intervals as arguments, we open the possibility that a proposition involving some predicate P might be neither true nor false over some interval t. In particular, consider a predicate P that is true during some subinterval of t and also false in some other subinterval of t. In this case, there are two ways we might interpret the negative proposition $\sim P(t)$. In the weak interpretation, $\sim P(t)$ is true if and only if it is not the case that P is true throughout interval t, and thus $\sim P(t)$ is true if P changes truth-values during t. In the strong interpretation of negation, $\sim P(t)$ is true if and only if P is false throughout t, and so neither $P(t)$ nor $\sim P(t)$ would be true in the above situation. Thus, a logic with only strong negation has truth gaps.

We take the weak interpretation of negation as the basic construct, as do Shoham [55] and Bacchus *et. al.* [10], to preserve a simple two-valued logic. Weak negation also seems to be the appropriate interpretation for the standard definition of implication. In particular, the formula $P(t) \supset Q(t')$ is typically defined as $\sim P(t) \vee Q(t')$. Since we want the implication to mean "whenever P is true over t, then Q is true over t'," this is best captured by the weak negation form of the equivalent formula. With weak negation, $\sim P(t) \vee Q(t')$ says that either P is not true throughout t (but might be true in some subintervals of t), or Q is true over t'. This seems the right interpretation. Of course, we can still make assertions equivalent to the strong negation. The fact that P is false throughout t can be expressed as

$$\forall t' \,.\, t' \sqsubseteq t \supset \sim P(t').$$

This is a common enough expression that we will introduce an abbreviation for the formula, namely $\neg P(t)$ (that is, the symbol "$\neg$" means strong negation). This way we obtain the notational convenience of strong negation while retaining the simpler semantics of a logic with no truth gaps.

There are several characteristics of propositions that allow them to be broadly classified based on their inferential properties. These distinctions were originally proposed by Vendler [59], and variants have been proposed under various names throughout linguistics, philosophy, and artificial intelligence ever since (*e.g.*, [45; 2; 16; 56]). For the most part we will not be concerned with all the distinctions considered by these authors. However one important property mentioned previously is homogeneity. Recall that a proposition is homogeneous if and only if when it holds over a time period t, it also holds over any period within t. In the current formulation, this property is defined by a family of axiom schemata, one for each arity of predicate. For all homogeneous predicates P of arity $n + 1$:

Homogeneity Axiom Schema

$$\forall x_i, t, t' \,.\, P(x_1, \ldots, x_n, t) \wedge t' \sqsubseteq t \supset P(x_1, \ldots, x_n, t').$$

All predicates will be homogeneous in what follows except when explicitly noted. The following useful theorem follows from the definition of strong negation and the homogeneity property, for any homogeneous predicate P:

DISJ $\forall t, t' \,.\, P(t) \wedge \neg P(t') \supset t \bowtie t'.$

That is, two intervals over which a predicate has different truth values (with respect to strong negation) must be disjoint.

There is still one aspect of the relationship between time and properties that remains to be addressed. This issue is crucially important to our subsequent development of mechanisms for reasoning about change. Thus far, we have not been concerned with whether the temporal structure is modelled on the real line or on some more restricted domain. However, once we start considering properties changing value in time, we are immediately confronted with the possibility of various sorts of infinities if the underlying model is the reals. For example, the definition of weak negation allows intervals over which $\sim P$ holds but for no subinterval of which does $\neg P$ hold. In such a case, we would have an infinite intermingling of P and $\sim P$ intervals.

While such strange situations are mathematically possible, they are clearly not what would expect from a commonsense theory. We therefore explicitly rule them out with a axiom schema of "discrete variation," adapted from Hamblin [28]:

Discrete Variation Axiom Schema

$$\forall t \,.\, \sim P(t) \supset \exists t' \,.\, t' \sqsubseteq t \wedge \neg P(t').$$

This states that any interval over which a property P is weakly false has (at least) a strongly false "core." Hamblin states that this schema, together with homogeneity, defines the "phenomenal" predicates, *i.e.*, those corresponding to natural phenomena.

It turns out that this is not in fact strong enough to draw all the commonsense conclusions one would expect.[1] For example, consider that we know of two intervals t and t' where $t \prec: t'$, such that $P(t)$ and $\neg P(t')$. A desirable (indeed fundamental) conclusion that we would like to draw from this premise is that there is a "transition point" between P and $\neg P$, *i.e.*,

$$\exists T, T' \,.\, P(T) \wedge \neg P(T') \wedge T : T' \wedge t' \sqsubseteq T'.$$

That is, there exist two intervals that meet where the property changes truth value, and these intervals are related to the original ones by inclusion on the left (there is a similar statement for the right). This conclusion, however, does not follow from discrete variation and homogeneity. The reason is that although the discrete variation schema rules out infinite intermingling, we can still have an infinite sequence of intervals of the *same* truth value telescoping towards a point, as in Zeno's Paradox.

[1] We are indebted to Nort Fowler for making this clear.

This problem is of course familiar to mathematicians from the theory of continuous functions. It led Galton [21] to introduce additional axioms to explicitly eliminate such telescoping intervals. More generally in AI, Davis [15] has pointed out several related problems with the relationship between continuous functions and AI theories of knowledge representation. Van Bentham [58] is the definitive treatment of these questions as regards temporal logic. For our part, we believe that it would be possible to import and adapt appropriate pieces of the theory of functions such that, with appropriate assumptions about, *e.g.*, continuity and density, we could derive the desired conclusions. The details are complicated, and in any event would mostly be a recapitulation of prior mathematical analysis. Not only are we not really qualified for the task, our purpose here is a commonsense theory, which the theory of functions certainly is not. We will therefore add the following "transition point" axiom schema to our temporal logic:

$$\textbf{TRPT} \quad \forall t, t' \,.\, P(t) \wedge \neg P(t') \wedge t \prec: t' \supset \\ \exists T, T' \,.\, P(T) \wedge \neg P(T') \wedge T : T' \wedge t' \sqsubseteq T'$$

What price have we paid for axiomatizing away this difficulty? Davis [15, p. 48] quotes Russell as stating that "the method of 'postulating' what we want has many advantages; they are the same as the advantages of theft over honest toil." In our own defense, we believe that our axiomatization is intended to produce a commonsense theory—one whose conclusions are those sanctioned by "common sense." The onus is therefore on the critics of the axioms to produce commonsense examples that violate the axioms. Davis argues that ruling out the use of some standard physical theories (as TRPT does) is too high a price to pay. Van Bentham, however, in his discussion of "linguistic" aspects of temporal logic (*i.e.*., using the logic in descriptions of the world), comments that "There is a lesson to be learnt here. Nature makes no jumps; but our linguistic descriptions can." [58, p. 229]

3.3 The Logic of Events

The logic developed thus far is still insufficient to conveniently capture many of the circumstances that we need to reason about. In particular, we need to introduce events as objects into the logic. There are many reasons for this, and the most important of these are discussed in the remainder of this section.

Davidson [14] argued that there are potentially unbounded qualifications that could be included in an event description. The issue of reifying events is not only an issue for representing natural language meaning, however. A sophisticated plan reasoning system also needs to represent and reason about events of similar complexity. In addition, in many forms of plan reasoning, the system must be able to distinguish events even though it does not have any information to distinguish them.

We use a representation that makes event variables the central component for organizing knowledge about events. In particular, events are divided into types, and each type defines a

set of role functions that specify the arguments to any particular event instance. The event of Jack lifting the ball onto the table at time $t1$ would be represented as

$$\exists e \,.\, \textsc{Lift}(e) \wedge \mathit{agent}(e) = \mathit{jack}34 \wedge \\ \mathit{dest}(e) = \mathit{table}5 \wedge \mathit{theme}(e) = \mathit{ball}26 \wedge \mathit{time}(e) = t1.$$

Event predicates will always be written in SMALL CAPS to distinguish them from other functions and predicates.[2]

This representation is somewhat verbose for presenting examples. When needed to make a point, the representation using only functions on events will be used. At other times, however, we will use the more standard predicate-argument notation as a convenient abbreviation. Thus, we will usually abbreviate the above formula as:

$$\exists e \,.\, \textsc{Lift}(\mathit{jack}34, \mathit{ball}26, \mathit{table}5, t1, e).$$

The arguments in this predicate-argument form will depend on the predicate, but the last two argument positions will always be the time of the event and the event instance, respectively. Finally note that event predicates are anti-homogeneous (that is, they hold over no subinterval of the time over which they hold) as discussed in Section 2.1.

Because of the representation based on role functions, an event instance uniquely defines all its arguments. This is important to remember when the predicate-argument abbreviation is used. In particular, if we asserted that both $\textsc{Lift}(a_1, b_1, c_1, t_1, e)$ and $\textsc{Lift}(a_2, b_2, c_2, t_2, e)$ were true, then this would entail that $a_1 = a_2$, $b_1 = b_2$, $c_1 = c_2$, and $t_1 = t_2$, a fact not obvious when using the predicate-argument form but clear from the functional form.

We will represent knowledge about events in several ways. The first is by defining necessary conditions on the event occurring. For instance, consider the event of one block being stacked on another by a robot. This could be described by an event predicate of form $\textsc{Stack}(x, y, t, e)$. Axioms then define the consequences of this event occurring. For instance, one might assert that whenever a stack event occurs, the first block is on the second block at the end of the action. In the predicate-argument notation, this could be written as

$$\forall x, y, t, e \,.\, \textsc{Stack}(x, y, t, e) \supset \exists t' \,.\, t : t' \wedge \mathit{On}(x, y, t').$$

In the functional form, the same axiom would be

$$\forall e \,.\, \textsc{Stack}(e) \supset \exists t' \,.\, \mathit{time}(e) : t' \wedge \mathit{On}(\mathit{block1}(e), \mathit{block2}(e), t').$$

[2]The logic containing reified events also allows one to discuss events that do not occur. In particular, asserting that an event instance exists does not necessarily entail that it occurred. A new predicate *Occurs* can be introduced and the assertion that Jack lifted the ball (at time $t1$) would be represented as $\exists e \,.\, \textsc{Lift}(\mathit{jack}34, \mathit{ball}26, t1, e) \wedge \mathit{Occurs}(e)$. Using an explicit *Occurs* predicate allows events to exist even if they do not actually occur, or could never occur. We will not need this expressive power in this paper so will proceed under the interpretation that only events that occur exist. Not having to put the *Occurs* predicate in each formula will simplify the presentation.

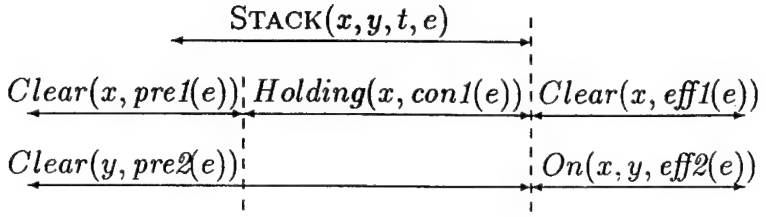


Figure 4: Necessary conditions for stacking x on y

Of course, there are many other necessary conditions in order for a stacking event to occur. For instance, we might say that the block moved (*block1*(e)) must be clear when the event starts, that the agent must be holding that block some time during the event (actually up to the end of the event), and that the other block (*block2*(e)) is clear just before the end of the event, and has *block1* on it immediately after the event completes. The event terminates at the time when *block1* is on *block2*. This information can be expressed directly using the temporal logic by the following axiom, also shown graphically in Figure 4:

$$\begin{aligned}
&\forall x, y, t, e \,.\, \textsc{Stack}(x, y, t, e) \supset \\
&\quad \exists j, k, l, m, n \,.\, Clear(x, j) \wedge Overlaps(j, t) \wedge \\
&\qquad Holding(x, k) \wedge Finishes(k, t) \wedge j : k \wedge \\
&\qquad Clear(x, l) \wedge t : l \wedge Clear(y, m) \wedge SameEnd(t, m) \wedge \\
&\qquad On(x, y, n) \wedge t : n.
\end{aligned}$$

A more useful form of this axiom for planning uses temporal functions on each event that define the structure of the temporal intervals needed for its definition. For example, for the class of stacking events, we need functions to produce times corresponding to the existential variables in the axiom given above. Using new function names, we might define the temporal structure of the stacking event as follows:

$$\begin{aligned}
&\forall t, e \,.\, \textsc{Stack}(t, e) \supset \\
&\quad Overlaps(pre1(e), t) \wedge Finishes(con1(e), t) \wedge pre1(e) : con1(e) \wedge \\
&\quad t : eff1(e) \wedge SameEnd(t, pre2(e)) \wedge t : eff2(e).
\end{aligned}$$

The temporal functions are named to informally suggest the three classes of conditions that arise in an event definition. The "preconditions"—conditions that must hold prior to the event's occurrence—have the prefix *pre*, the "effects"—conditions that must hold following the event—have the prefix *eff*, and the other conditions that must hold during the event have the prefix *con*. The temporal functions can be viewed as Skolem functions justified by the original axiom. The only aspect that will seem puzzling is that we will use the same Skolem functions in several axioms. This should be viewed as a notational convenience that allows a

large conjunctive axiom (with a single universally quantified event variable) to be presented as several smaller axioms. With this temporal structure defined for every stacking event, the axiom defining the necessary conditions for the event's occurrence now can be expressed as follows:

$$\begin{aligned}\forall x, y, t, e\,.\,&\textsc{Stack}(x, y, t, e) \supset \\ &\mathit{Clear}(x, \mathit{pre1}(e)) \wedge \mathit{Holding}(x, \mathit{con1}(e)) \wedge \mathit{Clear}(x, \mathit{eff1}(e)) \wedge \\ &\mathit{Clear}(y, \mathit{pre2}(e)) \wedge \mathit{On}(x, y, \mathit{eff2}(e)).\end{aligned}$$

Again, the combination of this axiom and the temporal structure axiom is shown graphically in Figure 4.

This axiom asserts what is true whenever a stacking event occurs, independent of the situation. Other knowledge about events is relevant only in certain situations (*i.e.*, the event has conditional effects). For instance, if the block being moved in a stacking action was initially on another block, then this other block becomes clear. This is expressed in the logic by the following axiom, which states that if block x was initially on another block z, then z becomes clear when x is moved:

$$\begin{aligned}\forall x, y, z, t, e, t'\,.\,&\textsc{Stack}(x, y, t, e) \wedge \mathit{On}(x, z, t') \wedge \mathit{Overlaps}(t', t) \supset \\ &\exists T, t''\,.\,t' \sqsubseteq T \wedge \mathit{On}(x, z, T) \wedge T : \mathit{con1}(e) \wedge \mathit{Clear}(z, t'') \wedge T : t''.\end{aligned}$$

Of course, if the only events we needed to represent were simple events such as occur in the blocks world, then the temporal logic would be overkill. But we are interested in much more realistic events, where we may have significant knowledge about how the world changes as the event occurs. As a quick example, consider the event of a cup filling up with water. At the end the cup is full, but while the event is occurring, the level of the water is continually increasing. Commonsense knowledge about this event is easily captured in the interval logic. For instance, the fact that the level of the water continues to rise throughout the event is captured by the following axiom:

$$\begin{aligned}\forall c, e, t\,.\,&\textsc{Fill}(c, t, e) \supset \\ &\forall t', t''\,.\,(t' \sqsubset t) \wedge (t'' \sqsubset t) \wedge (t' \prec: t'') \supset \\ &\mathit{level}(c, t') < \mathit{level}(c, t''),\end{aligned}$$

where $\mathit{level}(c, t)$ is now an interval-valued function that gives the range of level of the water in the cup over time t. This axiom captures the intuition that the water continually rises quite simply and directly without making a commitment to a continuous model of change. In order to draw conclusions from axioms such as this one, we would need to develop the mathematics of interval-valued functions in some detail. Van Bentham [58] presents several definitions of continuous functions in interval-based theories. Dealing with events that involve such "continuous change" is not the focus of this paper. however, and most of our examples will involve the very simple actions that are common in the literature on planning and reasoning about action.

Finally, some might wonder why the logic has no separate mechanism for describing processes, as was present in [2]. While this may ultimately be necessary, the existing formalism already supports a wide range of "process-like" predicates. In particular, you can define a property that is true only when an event is occurring. In the above example about the cup, we could define a predicate *CupFilling* as

$$\forall c, t \,.\, CupFilling(c,t) \equiv \exists t', e \,.\, t \sqsubseteq t' \wedge \textsc{Fill}(c,t',e)$$

Note that by this definition, *CupFilling* is a homogeneous predicate, as expected for properties. The place where problems arise with this simple approach is in dealing with the imperfective paradox. For instance, you might want to say that a person was crossing the street at some time t, even if they changed their mind at the last minute and went back to the original side. This problem has been studied extensively in the philosophical literature, but has not been a focus for our work as it does not seem to arise naturally in the planning domains we have studied.

3.4 The Logic of Actions

The representation of events described in the previous section does not adequately capture knowledge of causality. In particular, the formulas above do not state what properties are caused by the stacking action or what properties simply must be true whenever the action succeeds. This is the distinction that STRIPS makes between preconditions and effects. Intuitively, it is evident that the stacking action causes block A to be on block B in situations where both blocks are clear at the start of the action. Furthermore, the stacking action causes block B to become not clear while it does not affect the condition that block A is clear.

To encode such knowledge, we need to be able to reason about attempting to perform an action. To do this, we need to better define the distinction between events and actions. So far, we have only talked about events occurring. The assertion that Jack lifted the ball onto the table describes an event in which Jack performed some action that resulted in the ball being lifted onto the table. The action Jack performed, namely, the lifting, was realized by some set of motions that Jack performed in order to lift the ball. If Jack were a robot, the action would be the execution of a program that involved the correct control sequences given perceptual input. Thus, in a robot world the action corresponds to the program, whereas the event corresponds to a situation in which the program was executed successfully.

As noted in Section 2.1, for every action there is a corresponding event consisting of an agent performing that action. We will often exploit this by using the same names for events and actions. Thus the $\textsc{Stack}(x,y,t,e)$ predicate presented in the previous section might correspond to the action term $stack(x,y)$ that denotes a program where x and y correspond to the blocks being stacked. Of course, there are other events that do not involve actions. For example, natural forces (such as the wind blowing) result in events but do not involve action in the sense we are using it. Actions may be arbitrarily complex activities, and can

be decomposable into other less complex actions, which themselves may be decomposable, until a certain basic level of action is attained. The primitive actions are called the *basic* actions following [25].

The predicate Try is defined on programs, such that $Try(\pi, t)$ is true only if the program π is executed over time t.[3] As with event predicates, Try is anti-homogeneous. We can now assert an axiom defining the conditions sufficient to guarantee successful action attempts. In the case of stacking, whenever the agent tries to stack x on y starting in a situation where x and y are clear, then a stacking event occurs that is temporally constrained by the initial conditions:

$$\forall x, y, t, j, k \,.\, Try(stack(x,y), t) \wedge Clear(x, j) \wedge Overlaps(j, t) \wedge$$
$$Clear(y, k) \wedge SameEnd(k, t) \supset \exists e \,.\, \textsc{Stack}(x, y, t, e)$$

More realistic versions of this axiom might include duration constraints, and out theory can also have constraints on t such that $Try(\pi, t)$ is true (*e.g.*, that it is tried for long enough).

It is important to consider why the information about stacking is captured by two different sets of related axioms: one capturing the necessary conditions whenever a stacking event occurs (Section 3.3), and the other relating action attempts to event occurrences (above). This is because the two sets of axioms represent two very different sources of knowledge. The first defines knowledge about what the world is necessarily like whenever the event occurs successfully, while the second defines the abilities of the agent in causing events. In many situations, an agent may know the former but not the latter. For example, we all can recognize that the mechanic fixed our car, even if we have no idea what enabled the mechanic to do the job. Knowledge of this sort is also essential for much of natural language semantics, where many verbs are defined and used without the agent's knowing the necessary causal knowledge. Allen [2] discusses this at length.

To summarize the development thus far, we can partition our axioms describing events and actions into three broad categories:

EDEF Event definitions – These are axioms of the form

$$\forall e \,.\, E(e) \wedge \phi \supset \psi,$$

where E is an event-type predicate and ϕ and ψ contain no event predicates. The necessary conditions for stacking given in the previous section fall into this category.

ETRY Action definitions – These are axioms of the form

$$\forall t, \ldots \,.\, Try(\pi, t) \wedge \phi \supset \exists e \,.\, E(e) \wedge t \circ time(e) \wedge \psi,$$

[3] In previous work (*e.g.*, [3]), we included the event "caused" by attempting the action as an argument to the Try predicate. The problem with this is that we might want to categorize the "caused" event in several ways (*i.e.*, using several event predicates).

where again E is an event predicate. In this case, ψ represents constraints on e, and can involve other quantified variables. The symbol "o" stands for the temporal relation between the action and the event, typically "*Equal*" or "*Meets*", as discussed above.

EGEN Event generation axioms – These are of the form

$$\forall e \,.\, E(e) \wedge \phi \supset \exists e' \,.\, E'(e') \wedge \psi.$$

Again, E and E' are event-type predicates and ϕ and ψ represent constraints on the events. The classic example of event generation is represented in English using the "by" locution, for example, signaling a turn by waving one's arm, under appropriate conditions. More concrete examples will be presented in later sections.

Examples of all three classes of axioms will be presented in the next section when we describe application of the formalism to a set of problems from the literature on reasoning about action.

3.5 Discussion

Having introduced our representation of time, actions and events, we can now compare it to other formalisms. Note that this comparison is based on the expressiveness and simplicity of the logic for capturing commonsense knowledge. Dealing with prediction is a separate issue for most of these formalisms, and will be discussed at the end of the next section.

As discussed previously, most formalisms have not included an explicit model of time, but base their model on states or situations. To handle explicit temporal relations in the situation calculus, a function can be defined that maps a state or situation to a timepoint. In this way, the situation calculus defines a point-based branching time model. A duration function on actions can be defined that allows one to compute the time of the resulting situation given the initial situation [22]. Within the constructive models, however, the range of temporal reasoning that can be performed is severely limited. First, note that there are no situations defined during the time an action is executing. This means that you cannot assert anything about the world during an action execution. Rather, all effects of an action must be packaged into the resulting situation. As a consequence, all effects start simultaneously. Since situations correspond to time points, the formalism also has no mechanism for representing knowledge about how long an effect might hold, as time only moves forward as a result of an action. The only way around this seems to be to allow null actions that move time forward but have no effect, which is neither convenient or intuitive.

As a consequence, the only worlds easily represented by the constructive situation calculus are those that remain static except when the agent acts, and where nothing important happens while actions are being executed. Of course, if one abandons the requirement of a constructive theory, then more complicated scenarios can be represented. In fact, in some approaches (*e.g.*, [54]) even continuous change has been represented. The primary

mechanism that enables this, however, is the exploitation of the mapping from situations to times, and the introduction of situations that are not the result of any actions. Given these radical extensions, it is not clear to us what the advantage is in retaining the situation calculus as the underlying theory, rather than moving to the simpler, more direct, explicit temporal logic. Temporal logic gives properties, actions and events equal temporal status, allowing complex situations to be captured quite directly in a manner similar to how they are intuitively described.

There is no explicit representation of events in the situation calculus, which makes it difficult to represent knowledge of the actual world. For instance, it is not possible to directly assert that an event E will occur tomorrow, or even that the agent will perform an action A tomorrow. In the constructive situation calculus, one would have to quantify over all action sequences and ensure that the event or action was present. This does not seem workable, and recent attempts to represent the actual events generally abandon the constructive approach, and have to introduce an explicit new construct for events (*e.g.*, [43]). The resulting formalism is again closer to what we propose, and we question the advantage of retaining the original situation calculus framework.

The event calculus [32] has an explicit notion of event that corresponds more closely to our notion of events. This is not surprising as both this formalism and our own development of the temporal logic are based on intuitions about the close relationship between time and events. The event calculus does not seem to have a separate notion of actions, although formulas of the form $act(E, give(a, x, y))$ are used that suggest they might. As far as we can tell, however, this is only a syntactic mechanism for naming events. Reasoning in the event calculus is driven by the events that occur, as it is in our approach, but the relationship between the behaviors that an agent might perform and the events that the agent causes is not explored.

Closest to our representation is that described by McDermott [42]. He does distinguish between actions (which he calls tasks) and events, and uses an explicit model of time. As a result, he can define notions such as a successful action attempt, and explicitly reason about an agent's actions and what events they will cause. His representation of time has many similarities to the work that extends the situation calculus with a time line [49], and he explores the consequences of adding explicit time to such a model in depth. A significant difference between our approaches is that we use an interval-based logic, while McDermott uses a continuous point-based model. These differences have been discussed in detail elsewhere (*e.g.*, [58; 2; 56]).

4 Reasoning about Action in Simple Domains

As mentioned in the introduction, there are three major reasoning tasks we require the logic of events and actions to support: prediction, planning, and explanation. Of these, prediction is the most basic capability—given a description of the world and a set of actions and events

that will occur, what can we conclude about the past, present, and future? With a temporal logic, the initial world description might already contain some information about the past or the future, say some external events that will occur, or future actions that the agent knows it will perform. The prediction task reduces to predicting the effects of new actions and events that are posited to occur, and updating the world model accordingly. Neither planning nor explanation can be accomplished without a model of prediction. In planning, for instance, the task is to find a set of actions that will accomplish a given set of goals. This can be divided into two abstract tasks: generating a candidate set of actions, and evaluating whether the plan will succeed. This latter task is exactly the prediction task. Explanation can be similarly decomposed into generating a possible set of events that might explain the observations, and then verifying whether the events would actually cause the observed effects. Of course, any particular algorithm might not divide the reasoning into two explicit steps. In fact, most planning algorithms exploit a specific prediction model in order to suggest actions likely to produce a good plan, but all systems are based on some prediction model, which typically is based on the STRIPS assumptions (*e.g.*, [46; 13; 39]).

4.1 Frame Axioms and Explanation Closure

The problem is that the STRIPS assumptions do not hold in most realistic situations that one needs to reason about. The situation calculus and temporal logic-based approaches do not have to operate with such assumptions. As a result, these theories themselves make little commitment to how the state resulting from an action relates to the state before the action. Rather the properties of the resulting state must be specified axiomatically, and the frame problem involves how best to specify these properties. The original proposal for the situation calculus [41] was to use *frame axioms*, which explicitly stated which properties are not changed by the actions. Explicit frame axioms have come under criticism, primarily because there are too many of them, both to write down explicitly and to reason with efficiently.

There are several ways to try to overcome this problem. Many researchers abandon frame axioms altogether, and have built models that use persistence or inertia assumptions (*e.g.*, [34; 56]). These approaches assume that all changes caused by an action are specified, and every property not asserted to change does not change. This technique has much to recommend it, as it eliminates the need to enumerate frame axioms, but in its simple form it tends to be too strong. In particular, if there is uncertainty as to whether a property might change or not, techniques based on this approach will often incorrectly assume that the change does not occur. Other approaches work instead by minimizing event or action occurrences. Properties are assumed to change only as a result of events defined in the representation, and logically unnecessary events do not occur (*e.g.*, [24; 23; 44]). These approaches show more promise at handling more complex situations and have many similarities to our work.

The approach we take, however, retains the flavor of explicit frame axioms. Rather than specifying for each action whether each property changes or not. however, one specifies for each property what events can change it. The problem of reasoning about changes then reduces to the problem of reasoning about what events may or may not have occurred. This technique is called the *explanation closure* approach and was proposed by Haas [27] and Schubert [52]. Schubert has shown that such a technique can dramatically reduce the number of frame axioms required to produce a workable set of axioms for a problem. Of course, assumptions must still be made. As in other approaches, we assume that unnecessary events do not occur, but this is specified axiomatically rather than being built into the semantic model. This has several advantages. Most importantly, the resulting axioms can be interpreted with the standard semantics of the first-order predicate calculus, meaning that there are well-defined notions of entailment and proof. We can show that our representation handles a particular class of examples by showing a proof of the desired consequences, without needing to appeal to model-theoretic arguments in a non-standard semantics. In addition, various forms of uncertainty are handled using the standard methods in logic with disjunction and existential quantification. Finally, the assumptions that are made appear explicitly as axioms in the system. While not exploited in this paper, this allows for explicit reasoning about the assumptions underlying a line of reasoning (*cf.* [19; 18; 3; 17]).

If you are willing to reinstate strong assumptions about the domain, roughly equivalent to the STRIPS assumptions, then Reiter [50] has shown that the explanation closure axioms can be computed automatically using predicate completion techniques. But this close correspondence breaks down in more complex domains. Schubert [52] argues that explanation closure is distinct from predicate completion or biconditionalization, and thus that the axioms cannot be generated automatically. Specifically, he argues that any automatic procedure will produce axioms that are too strong in cases where knowledge of the world is incomplete and actions may have conditional effects. In addition, he argues that explanation closure axioms have "epistemic content" and are independently motivated by other problems such as language understanding. As such, they form a crucial body of knowledge necessary for many commonsense reasoning tasks.

The development to follow in the rest of this section does not demonstrate the full power of our approach, as it does not consider simultaneous actions and external events. This will facilitate the comparison of our work to other approaches. More complex cases are considered in detail in [5]. Because of the simple nature of the problems, the closure axioms can be classified into two classes, corresponding to two assumptions: no properties change unless explicitly changed by an event occurring, and no events occur except as the result of the actions. Although these are difficult to precisely describe schematically, they look something like the following following:

EXCP (Strong Closure on Properties) Every property change results from the occurrence of an instance of one of the event types defined in the axioms. These axioms are of the form:

$$\forall t, t' \,.\, P(t) \wedge \neg P(t') \wedge t : t' \supset \exists e \,.\, (E_1(e) \vee E_2(e) \vee \ldots) \wedge \mathit{time}(e) : t',$$

where the E_i are the event-type predicates that (possibly) affect the truth value of P. These axioms are derived from the event definition axioms (EDEF).

EXCE (Strong Closure on Events) Every event that occurs does so as a result of some action being attempted, possibly indirectly via event generation. These axioms are of the form: "$\forall e \,.\, E(e) \supset \phi_1 \vee \phi_2 \vee \ldots$," where ϕ_i is either of the form

$$\exists t \,.\, Try(\pi, t) \wedge t \circ time(e) \wedge \psi,$$

for π an action term and "$\circ$" a temporal relation (typically "$=$"), or of the form

$$\exists e' \,.\, E'(e') \wedge time(e) \circ time(e') \wedge \psi,$$

for E' an event-type predicate. These axioms can often be derived from the action definition (ETRY) and event generation (EGEN) axioms.

4.2 Example

By way of illustrating the application of the logic and the explanation closure technique, we now present the formalizations of one of the standard problems from the literature on reasoning about action, taken from the Sandewall Test Suite [51]. In [5], we consider a much wider range of problems from the test suite.

Consider our formulation of the basic Yale Shooting Problem (YSP). The scenario involves a hapless victim, Fred, who apparently sits idly by while an adversary loads a gun, waits some period of time, and shoots (at Fred). The question is whether Fred is dead or not, or rather whether an axiomatization of the problem in some logical system allows the conclusion that Fred is dead to be derived. This simple problem has generated an extensive body of literature (*cf.* [12]). In most cases, the issue is that while the loadedness of the gun ought to persist through the waiting (and then the shooting succeeds in killing Fred), Fred's "aliveness" ought not to persist through the shooting (indeed, cannot, without risk of inconsistency) although it should persist through the loading and waiting. On the other hand, there might be "non-standard" models where the gun somehow becomes unloaded, in which case the shooting would fail to kill Fred.

Figure 5 shows the axioms for the YSP in our logic; they are depicted graphically in Figure 6. Recall that the EDEF axioms describe the necessary conditions for event occurrence and the ETRY axioms describe the sufficient conditions relating program executions (actions) and events. For loading, these are both almost trivial: any attempt to load will cause a LOAD event (ETRY1) and this will entail that the gun is loaded immediately thereafter (EDEF1). A successful SHOOT event, on the other hand, entails that the gun be loaded beforehand, unloaded afterwards, and also that Fred be dead afterwards (EDEF2). Attempting the shooting action will be successful in causing such an event if the gun was loaded at the time of the shooting (ETRY2). Of course, the example axioms are very simple—there are only temporal roles and there are no generation axioms (yet).

EDEF1 $\forall e\,.\,\textsc{Load}(e) \supset \mathit{Loaded}(\mathit{eff1}(e)) \wedge \mathit{time}(e) : \mathit{eff1}(e)$

EDEF2 $\forall e\,.\,\textsc{Shoot}(e) \supset$
$\mathit{Loaded}(\mathit{pre1}(e)) \wedge \mathit{SameEnd}(\mathit{pre1}(e), \mathit{time}(e)) \wedge$
$\neg \mathit{Loaded}(\mathit{eff1}(e)) \wedge \mathit{time}(e) : \mathit{eff1}(e) \wedge$
$\neg \mathit{Alive}(\mathit{eff2}(e)) \wedge \mathit{time}(e) : \mathit{eff2}(e)$

ETRY1 $\forall t\,.\,\mathit{Try}(\mathit{load}, t) \supset \exists e\,.\,\textsc{Load}(t, e)$

ETRY2 $\forall t\,.\,\mathit{Try}(\mathit{shoot}, t) \wedge \mathit{Loaded}(t) \supset \exists e\,.\,\textsc{Shoot}(t, e)$

Figure 5: Basic axioms for the Yale Shooting Problem

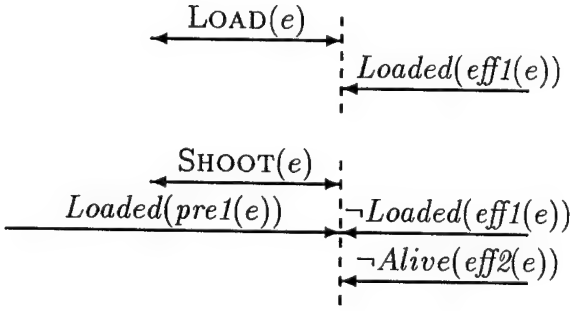


Figure 6: The LOAD and SHOOT event definitions

EXCP1 $\forall t, t' \,.\, \neg \mathit{Loaded}(t) \wedge \mathit{Loaded}(t') \wedge t : t' \supset$
$\exists e \,.\, \textsc{Load}(e) \wedge \mathit{time}(e) : t'$

EXCP2a $\forall t, t' \,.\, \mathit{Loaded}(t) \wedge \neg \mathit{Loaded}(t') \wedge t : t' \supset$
$\exists e \,.\, \textsc{Shoot}(e) \wedge \mathit{time}(e) : t'$

EXCP2b $\forall t, t' \,.\, \mathit{Alive}(t) \wedge \neg \mathit{Alive}(t') \wedge t : t' \supset$
$\exists e \,.\, \textsc{Shoot}(e) \wedge \mathit{time}(e) : t'$

EXCE1 $\forall e \,.\, \textsc{Load}(e) \supset \exists t \,.\, \mathit{Try}(\mathit{load}, t) \wedge t = \mathit{time}(e)$

EXCE2 $\forall e \,.\, \textsc{Shoot}(e) \supset \exists t \,.\, \mathit{Try}(\mathit{shoot}, t) \wedge t = \mathit{time}(e)$

Figure 7: Explanation closure axioms for the YSP

Figure 7 gives the explanation closure axioms corresponding to the event and action axioms in Figure 5. Given the simple actions and strong assumptions made in the problem formulation, an automatic procedure could probably devised to generate them. But notice that the closure axioms are not simply a strengthening of the implication to a biconditional. In particular, the EXCP2b axiom does not say that every SHOOT event causes a transition from alive to dead, just that every such transition is the result of a SHOOT. This would allow one to shoot things that are already dead, for example.

We can now present the axiomatization of the YSP scenario. Initially a turkey (taking the role of Fred) is alive and the gun is not loaded. The agent loads the gun, waits, and shoots. We are to show that the turkey is dead sometime after the shooting. At issue is the persistence of loadedness through the waiting and the non-persistence of aliveness through the shooting. The following axioms describe the scenario:

AX0 $t0 \prec t1 \prec t2 \prec t3$

AX1 $\mathit{Alive}(t0)$

AX2 $\neg \mathit{Loaded}(t0)$

AX3 $\mathit{Try}(\mathit{load}, t1)$

AX4 $\mathit{Try}(\mathit{shoot}, t3)$

AX5 $\forall a, t \,.\, \mathit{Try}(a, t) \equiv (a = \mathit{load} \wedge t = t1) \vee (a = \mathit{shoot} \wedge t = t3)$

The final axiom is the important assumption that we attempt only the given actions. With the EXCE axioms, this ensures that "nothing else happens," which drives the explanation closure method. Note that the formalism does not require such an assumption. Rather, we are making explicit an aspect of the scenario that is usually implicit and accomplished via some form of semantic minimization.

To Prove: $\exists t \,.\, \neg \mathit{Alive}(t) \wedge t3 \prec : t$

Since there are no preconditions for loading, AX3 and ETRY1 give

$$\exists e_1 \,.\, \textsc{Load}(t1, e_1),$$

then EDEF1 gives

$$\mathit{Loaded}(\mathit{eff1}(e_1)) \wedge t1 : \mathit{eff1}(e_1).$$

That is, the loading succeeds, and the gun is loaded, at least immediately afterwards. Now, suppose that

$$\mathit{Loaded}(t3) \qquad (*)$$

that is, the gun remains loaded until the shooting is attempted. In that case, AX4 and ETRY2 give $\exists e_3 \,.\, \textsc{Shoot}(t3, e_3)$, and then EDEF2 gives

$$\neg \mathit{Alive}(\mathit{eff2}(e_3)) \wedge t3 : \mathit{eff2}(e_3).$$

That is, the shooting also succeeds and the turkey is dead afterwards, as required.

To show the persistence $(*)$, suppose otherwise that $\neg \mathit{Loaded}(t3)$. The interval temporal logic theorem DISJ (Section 3.2) gives $\mathit{eff1}(e_1) \bowtie t3$, then AX0 and the fact that $t1 : \mathit{eff1}(e_1)$ give $\mathit{eff1}(e_1) \prec\!: t3$. We apply interval temporal logic axiom TRPT to get

$$\exists T, T' \,.\, \mathit{Loaded}(T) \wedge \neg \mathit{Loaded}(T') \wedge T : T' \wedge t3 \sqsubseteq T'.$$

Then, EXCP2a gives

$$\exists e \,.\, \textsc{Shoot}(e) \wedge \mathit{time}(e) : T'.$$

That is, if the gun became unloaded it must have been because of a shooting. Note that $\mathit{time}(e) \prec\!: t3$ since $t3 \sqsubseteq T'$ and $\mathit{time}(e) : T'$. Then from EXCE2 we get

$$\exists t \,.\, \mathit{Try}(\mathit{shoot}, t) \wedge t = \mathit{time}(e),$$

i.e., if there was a shooting then someone must have tried to shoot. Since $t = \mathit{time}(e) \prec\!: t3$, we have $t \neq t3$, contradicting AX5. □

4.3 Discussion

Work on the frame problem investigates methods of making assumptions about the world to predict the likely consequences of actions. There are therefore two separable issues to consider, which have been called the "epistemological" and "computational" aspects of the problem (*cf.* [31], but also [41] regarding epistemological and heuristic adequacy). The epistemological aspect concerns what assumptions one makes about the world, while the computational aspect concerns how to compute and use these assumptions in the formalism. For example, it is an epistemological issue whether to make assumptions about property changes as in the persistence-based approaches, or to make assumptions about event occurrences. On the other hand, it is a computational issue whether to use minimization

techniques in the model theory to implement assumptions, or to use explicit axioms, as in the original situation calculus or with the explanation closure technique. We will discuss each of these issues separately, although they are, of course, closely related.

Our approach is based on making assumptions about event occurrence, both events caused by actions and external events. In this way, it is similar to work by Morgenstern and Stein [44], Haas [27], Schubert [52], and Kowlaski and Sergot [32]. In such approaches, properties do not change unless there is some event that causes them to change. Approaches based on STRIPS-style representations can be viewed in this way as well, except that the assumptions are not a separable part of the formalism. Rather, the representation only makes sense when these assumptions are made. The other major approach focuses on minimizing property change, with additional constraints based on the temporal ordering of properties (*e.g.*, [56; 30]) or minimizing causal relationships (*e.g.*, [34]). Sandewall [51] examines the advantages and limitations of each of these approaches in detail. Most of the approaches that minimize property change cannot handle Sandewall's test suite of problems, let alone be extended to handle external events and simultaneous actions. We find that basing the assumptions on events leads to much more intuitive characterization of problems, where each statement in the logic is closely related to an intuitive fact about the world. In addition, in [5] we show how this approach naturally handles a wide range of more complex problems.

The second issue is what mechanism is used to make the assumptions. There are two main approaches here: explicitly adding axioms that encode the assumptions (*e.g.*, [26; 52]) or using a nonmonotonic model theory that defines a new notion of entailment that includes the assumptions (*e.g.*, [40; 56; 11; 51]). Of course, the work on circumscription shows that model-theoretic techniques always have an equivalent axiomatic formulation, although it may require going beyond standard first-order logic. This equivalence suggests that there is really a continuum of approaches here. Everyone must make assumptions. Some prefer to pack it all in the model theory, others use axioms to capture much of the complexity and use only simple minimization assumptions, while others prefer to encode everything in axioms. Many of the issues come down to ease of formalization, an issue that is bound to vary from researcher to researcher. The reason we prefer explanation closure axioms is that they give us a very flexible system that is easily extended to handle complex issues in representing actions. In addition, the resulting representation is a standard first-order logic, so it is relatively straightforward to tell if certain consequences follow from the axioms. The same cannot be said for approaches based on specialized reasoners or specialized semantic theories that solve one problem at the expense of the others. Schubert [52] provides an excellent discussion of the advantages of the explanation closure approach.

5 External Events and Simultaneous Actions

We now turn our attention to more interesting and complex cases of reasoning about action. This section gives a brief overview of how our formalism handles external events and simultaneous actions—two problems that are beyond the scope of many representations. There

is not the space for detailed examples and dicsussion. The interested reader should see [5] for more detail.

5.1 External Events

External events arise for different reasons. Some simply occur as a result of natural forces in the world, whereas others are set into motion by the action of the planning agent. Some external events are predictable—we know, for instance, that the sun will rise tomorrow, and that many of the central events of a normal workday will occur. But even when we are certain that a particular event will occur, we are often still uncertain as to when it will occur. Most of the time, for instance, I can only roughly estimate when the sun will rise, but this uncertainty doesn't generally affect my plans. Rather, I know that by a certain time, say 6AM at this time of year, the sun will be up. In most realistic applications, the planning agent will not have complete knowledge of the world and its causal structure, and thus there will be many external events for which it cannot reliably predict whether they will occur or not. Assuming that such events do not occur will lead to highly optimistic plans. Rather, an agent should be able to plan given the uncertainty of external events.

The logic presented in Section 3 already provides the formalism for representing external events. In fact, the only way to distinguish external events from events caused by the agent's acting is by the presence of an axiom involving the predicate $Try(\pi, t)$, that states that the event was caused by an action attempt. But there is no requirement that all events have axioms relating them to action attempts.

The complications arise in characterizing the appropriate explanation closure axioms. In particular, if one makes the strong event closure assumptions (as in the previous section), that all events are ultimately caused by some action, then external events caused solely by natural forces or other agents cannot be represented. Of course, the explanation closure approach doesn't require such a strong assumption. If fact, we could simply not have closure axioms for some event classes that are caused by natural forces. But this would then allow such events to interfere with the prediction process at every possible opportunity, as you could never prove that the event didn't occur. While this might be the right result in some situations, usually external events are more constrained.

There are two orthogonal aspects of external events that affect the representation. The first relates to uncertainty about whether the event will occur. A non-probabilistic logic can only easily represent the two extremes on this scale: either the event definitely will occur, or it may or may not occur. The second relates to the conditions for an event's occurrence. Again, there is a continuum here between a complete lack of constraints on when the event may occur, and cases where the event will only occur under specific conditions, such as being caused by the agent's actions. Somewhere in the middle of this scale would be events that may only occur within a certain time period, but that are independent of the agent's actions. There are three simple combinations that appear frequently in many domains. These are:

1. Triggered events: The external event will not occur unless it is specifically triggered by an agent's actions (*e.g.*, the microwave will heat my coffee only if someone presses the "on" button).

2. Definite events: The external event will definitely occur, independent of the agent's actions, although the exact time may be uncertain (*e.g.*, the sun will rise between 5:30 and 6:30AM).

3. Spontaneous events: The external event may or may not occur during some time period (*e.g.*, I might win the lottery tonight).

Our representation can also handle more complex cases, such as triggered spontaneous events, which only become possible as a result of some action by the agent. We only have space here to consider a simple example of a triggered event. In [5], we present examples showing how definite and spontaneous events are handled.

Triggered events are actually already handled by the development so far. In fact, all events in the last section can be viewed as triggered events that characterize the agent's actions. The same approach can be used to handle other triggered external events.

Sandewall's Hiding Turkey Scenario is quite naturally handled as a simple case of reasoning about triggered external events. In this setting, the turkey may or not be deaf. If it is not deaf, then it will go into hiding when the gun is loaded, and thus escape death from the shooting. That is, we are given information about how the turkey will respond to a certain situation, namely that it will hide if it hears the gun being loaded. We must conclude that after the shooting, either the turkey is alive and not deaf, or it is dead.

Many formalizations of this problem treat the turkey hiding as an conditional effect of loading the gun rather than as an external event. If this approach worked for all triggered external events, then it might be justified. But unfortunately, it only works in the simplest of cases, namely when the event caused has no structure or temporal complexity, and so can be characterized as a static effect. For example, consider a situation where the only place the turkey might hide is in a box, and the box has a door that can be closed by the agent. A natural formalization of this would include the following facts:

1. If the turkey is not deaf, it will try to hide when the gun is loaded.

2. If the box door is open when the turkey tries to hide, it will be hidden in 30 seconds.

You could capture some of this situation by adding a new effect rule about loading that if the box door is open, the turkey will hide. But it is not clear how you might encode the fact that the turkey is not hidden for 30 seconds, so if you shoot quickly you could still hit it. As the scenario becomes more complicated, more and more information would have to be packed into the load action rather than the hiding event. Even if this could be done, it would very unintuitive, as it would not allow you to reason about the turkey hiding except in this one case when the agent loads the gun. What if the turkey may hide for other reasons

as well, or might spontaneously decide to hide, or always hides between 6PM and 7PM? Clearly, hiding is just as valid an event as any other event (such as loading), and requires similar treatment.

Given all this, let's return to the simple example as originally formulated. Taking $Deaf$ to be an atemporal predicate, the triggering of hiding by loading is captured with a conditional generation axiom:

EGEN3 $\forall e\,.\,\textsc{Load}(e) \wedge \neg Deaf \supset \exists e'\,.\,\textsc{Hide}(e') \wedge time(e) = time(e')$

To capture the simplicity of the original problem and other solutions in the literature, we assume that the hiding is simultaneous with the loading. The more natural formulation would complicate the proof as we would need to reason about whether the agent could shoot while the turkey was trying to hide.

Next, we need to add axioms for hiding events to the YSP axioms from Figure 5. These are:

EDEF3 $\forall e\,.\,\textsc{Hide}(e) \supset Hidden(\mathit{eff1}(e)) \wedge time(e) : \mathit{eff1}(e)$

EXCP3 $\forall t, t'\,.\,\neg Hidden(t) \wedge Hidden(t') \wedge t : t' \supset$
$\exists e\,.\,\textsc{Hide}(e) \wedge time(e) : t'$

EXCE3 $\forall e\,.\,\textsc{Hide}(e) \supset \neg Deaf \wedge \exists e'\,.\,\textsc{Load}(e') \wedge time(e') = time(e)$

The EDEF3 axiom simply states that a HIDE event results in the turkey being hidden. EXCP3 is a closure axiom that says that things become hidden only as a result of a HIDE event occurring. Closure axiom EXCE3 is justified by the important information implicit in the problem statement that the turkey does not hide unless it hears the gun being loaded.

We then need to modify the definition of shooting (EDEF2) so that it only kills the turkey if it's not hidden:

EDEF2a $\forall e\,.\,\textsc{Shoot}(e) \supset$
$Loaded(\mathit{pre1}(e)) \wedge SameEnd(\mathit{pre1}(e), time(e)) \wedge$
$\neg Loaded(\mathit{eff1}(e)) \wedge time(e) : \mathit{eff1}(e)$

EDEF2b $\forall e\,.\,\textsc{Shoot}(e) \wedge \neg Hidden(time(e)) \supset$
$\neg Alive(\mathit{eff2}(e)) \wedge time(e) : \mathit{eff2}(e)$

Note that this characterization of a SHOOT event means "the bullet leaving the gun," presumably in the direction of the turkey. It does not mean "kill the turkey," which is rather a conditional effect of the shooting (EDEF2b). The action *shoot* is "pulling the trigger" on this account.

The axioms describing the problem are then as follows (the YSP axioms plus new axioms AXH1 and AXH2):

AX0 $t0 \prec t1 \prec t2 \prec t3$

AX1 $Alive(t0)$

AX2 $\neg Loaded(t0)$

AXH1 $\neg Hidden(t0)$

AXH2 $\forall t, t' \,.\, Hidden(t) \wedge t \prec: t' \supset Hidden(t')$

AX3 $Try(load, t1)$

AX4 $Try(shoot, t3)$

AX5 $\forall a, t \,.\, Try(a, t) \equiv (a = load \wedge t = t1) \vee (a = shoot \wedge t = t3)$

Note that axiom AXH2 simply states that once the turkey is hidden it remains hiding forever. A more realistic axiomatization might allow the possibly of an UNHIDE event, and then provide explanation closure axioms for when it might occur. The details are irrelevant for purposes of the example.

To Prove: $\exists t \,.\, t3 \prec: t \wedge ((Deaf \wedge \neg Alive(t)) \vee (\neg Deaf \wedge Alive(t)))$

The following is a sketch of the proof, since many of the details are similar to the previous proof.

(a) Suppose $Deaf$, so we need to show that the turkey is killed. The issue is whether the turkey is hiding at the time of the shooting, *i.e.*, whether $Hidden(t3)$. Suppose it is. Then, from AX0, AXH1, and TRPT we get

$$\exists T, T' \,.\, \neg Hidden(T) \wedge Hidden(T') \wedge T : T' \wedge t3 \sqsubseteq T'.$$

Then EXCP3 gives:

$$\exists e \,.\, \textsc{Hide}(e) \wedge time(e) : T'.$$

But then EXCE3 gives $\neg Deaf$, a contradiction, since the turkey only hides if it is not deaf. Thus $\neg Hidden(t3)$. Since the turkey is not hidden when the shooting occurs, axiom EDEF2b allows us to conclude that the turkey is killed afterwards, as required assuming that reincarnation is ruled out axiomatically.

(b) Suppose instead $\neg Deaf$, so we need to show that the turkey is not killed. AX3 and ETRY1 give

$$\exists e \,.\, \textsc{Load}(e) \wedge time(e) = t1$$

and then EGEN3 gives

$$\exists e' \,.\, \textsc{Hide}(e') \wedge time(e') = t1.$$

That is, the loading succeeds and the turkey hides, since it is not deaf. This is consistent with EXCE3, and then EDEF3 yields $Hidden(\mathit{eff1}(e')) \wedge t1 : \mathit{eff1}(e')$. This persists indefinitely (AXH2), in particular until $\mathit{pre2}(e_3)$, so the shooting fails to kill the turkey (EDEF2b doesn't

apply). Thus *Alive* persists indefinitely starting at $t0$ (EXCP2b, AX5), from which we can derive the desired result. □

Triggered events are very useful for characterizing many domains. For example, whenever you press a button to start a machine, you trigger an event. Our formalism allows you to define the behavior of the machine using the full power of the language to describe events. Thus, pressing one button could trigger a complex sequence of events that greatly affects the domain for extended periods of time. A simple example could involve making a simple meal of reheated pizza. The plan is to put the pizza in the microwave oven and then press the start button. While it is cooking, you take a beer out of the refrigerator and get out the dishes. When the microwave beeps, you take out the pizza and eat. The fact that the microwave is running could constrain other actions you can perform during that time. For instance, you couldn't also reheat some coffee using the microwave, as it is in use. Or if the electric service is limited, you might not be able to run the toaster oven at the same time without blowing a fuse. As simple as this scenario sounds, representing it is beyond the capabilities of most formalisms.

5.2 Discussion

Most formalisms prohibit external events. For example, action definitions in STRIPS-based systems must encode all changes in the world, so external events are fundamentally banned from analysis. Similarly, the constructive situation calculus has the same problem—since there is no way to express information about what the state of the world is while the action is happening, there is no mechanism for allowing an event to occur while an action is being performed. Lifschitz and Rabinov [35] present a limited mechanism to handle this by allowing "miracles" to occur while an action is performed to explain why its effects are not as expected. But this does not allow external events to occur independently of actions, nor does it give events the first-class status required to represent complex situations. In addition, by minimizing the number of miracles, the approach makes it difficult to handle uncertainty about whether some external event will occur or not. Another possibility would be to encode external events as pseudo-actions so that they can be used in the result function, although they are clearly different from normal actions (*e.g.*, the agent couldn't plan to execute them).

Work based on the situation calculus that addresses external events typically rejects the constraints of the constructive approach and uses the more general formalism. In these approaches situations are arbitrary states of the world, not necessarily related to a particular action sequence. In addition, situations can be associated with times from a time line and one can quantify over situations. With this, one can introduce an *Occurs* predicate that asserts that a particular action occurs at a specific time and that is defined by an axiom that quantifies over all situations at that time (*e.g.*, [49; 43]). This development moves the formalism closer to what we are proposing.

Ultimately, the main difference between our approach and these extended situation calculus representations with explicit time will probably be one of approach. We start from

a representation of the actual world (or an agent's beliefs about the world), and must introduce mechanisms to allow reasoning about the effects of possible actions. The situation calculus starts from a representation based on possible futures based on the actions, and must introduce mechanisms for dealing with information about the actual world.

5.3 Simultaneous Action

There are several levels of difficulty in dealing with simultaneous actions. The model described so far already handles some cases of simultaneous actions, namely

1. When two actions cannot occur simultaneously;
2. When they occur simultaneously and are independent of each other; and
3. When they together have additional effects that neither one would have individually.

The more difficult case is when two actions partially or conditionally interfere. After first describing how the basic approach handles the three cases above, we will discuss some approaches to reasoning about interference.

We prefer to use a more complex domain than that of the YSP to illustrate these cases. The TRAINS domain is a transportation and manufacturing domain developed for use in the TRAINS project [8; 9]. The goal of the project is to build an intelligent planning assistant that is conversationally proficient in natural language. The domain involves several cities connected by rail links, warehouses and factories at various cities, and engines, boxcars, *etc.*., to transport goods between them. In this domain, interacting simultaneous actions and external events are unavoidable aspects of the domain.

When actions are independent of each other, the existing formalism does exactly the right thing. The effect of the two actions is simply the sum of the effects of actions individually, a result you get directly from the axioms. When there are simple interactions, such as mutual exclusion of actions under certain circumstances, this behavior typically naturally falls out of a reasonable axiomatization of the domain.

Consider the simple TRAINS domain scenario shown in Figure 8. Now consider simultaneously sending engine N1 to Corning via Bath and sending N2 via Avon to Dansville. A reasonable axiomatization of the move actions will allow us to conclude that both get to their destinations. Now let's assume that, in this domain, only one train can be on a track at a time, and that this constraint is captured by an axiom. For instance, we might require that a track be clear for an engine to successfully move along it. With such an axiomatization, if we try to send N1 to Corning via Avon and N2 to Dansville via Avon simultaneously, then one of the move actions will not succeed since one of the tracks along the route will not be clear when needed. More details are given in [5].

Notice that the fact that the actions were attempted is still true. In general, we do not limit action attempts in any way, reflecting a belief that an agent may attempt most

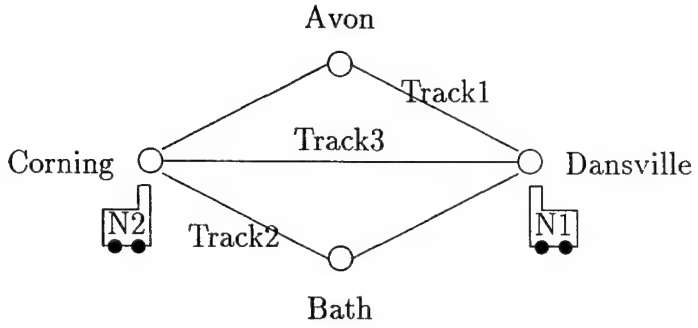


Figure 8: Example TRAINS domain map

actions at any time. If the conditions are not appropriate, the actions simply won't cause the desired events. Of course, we could add additional axioms describing what happens when action attempts interact. For instance, we could add a dramatic axiom that asserts that if two move actions are attempted simultaneously on the same track, the engines crash. More realistically, we might predict a deadlock situation, or predict that one will wait while the other passes. The point here is that the temporal logic provides a fairly natural way to axiomatize the knowledge in this domain so as to capture the desired behavior.

Another form of interaction between simultaneous action attempts are resource conflicts. For instance, in the TRAINS domain, an engine requires fuel to make trips. If two engines are planning trips but there is only enough fuel at the station for one, then only one engine can succeed. Many of these cases require reasoning about quantities, which would introduce many complications not necessary to make the points in this paper. But some resources can be modeled as unitary—either you have the resource or you don't. In these cases, the techniques used above can be adapted to resources. In fact, you can view the track in the previous example as a resource required by an engine to make a trip, and the problem where the engines try to use the same track as a resource conflict.

Other constraints on simultaneous action arise because of limitations on the part of the agent. While often such restrictions can be cast in terms of resource conflicts, the temporal logic also allows specific axioms about action co-occurrence. For instance, say a engineer cannot try to couple one car and decouple another at the same time. This constraint can be simply captured by the axiom:

$$\forall n, c, c', t, t' \,.\, Try(couple(n, c), t) \wedge Try(decouple(n, c'), t') \supset t \bowtie t'$$

This axiom would make a set of assertions where an agent simultaneously attempted to couple one car and decouple another inconsistent. Axioms like this can be very useful in defining simple planning systems similar to non-linear planners (*e.g.*, [7]).

It is also easy to define additional synergistic effects that occur when two actions occur simultaneously. For instance, you can define events that occur only when several actions are

performed simultaneously. Thus, if one agent lifts one end of a piano while another agent lifts the other end, then the entire piano is lifted off the floor. Such situations are easily described in our representation. For example, assuming axioms for a *lift* action that causes LIFT events, we could specify how the simultaneous lifting of both ends of the piano results in the piano being lifted using a generation axiom such as:

$$\forall p, t, e1, e2 \,.\, \text{LIFT}(left(p), t, e1) \wedge \text{LIFT}(right(p), t, e2) \supset \exists e3 \,.\, \text{LIFT}(p, t, e3).$$

Complications in handling simultaneous actions arise when they partially interfere with each other, or interfere under certain conditions. Note that with our view of events as classifications of patterns of change, it makes no sense to talk about events interfering with each other. The same world cannot be characterized by two events whose definitions are mutually inconsistent. Interference makes sense only when talking about actions. For instance, if an agent is trying to open a door, then it is performing a *push* action hoping to cause an OPENDOOR event. If, simultaneously, another agent pushes on the door the other way, then the first agent will still have performed the *push* action, but the OPENDOOR event will not have occurred. Thus, we say that the two *push* actions interfered with each other. In this case, they cancelled each other out. Of course, action attempts might also be affected by external events that are occurring simultaneously.

When an action interacts with other events that are occurring, the interaction can be treated much the same as we handled conditional effects of action attempts when different properties hold. This technique formalizes the problem in terms of events, and the different effects result from synergistic interaction between events. For example, consider a situation where we want to load a boxcar from an automatic hopper. If we push the start button, the hopper tips and dumps its cargo into the boxcar. This, of course, requires that the boxcar be in the appropriate location under the hopper. If the boxcar was not under the hopper to start with, it is clear that the loading of the car would not occur although the hopper would still be emptied. The more interesting case occurs when the boxcar is initially under the hopper, but then moves away while being loaded. Intuitively, we say that the moving of the boxcar interferes with the loading of the car. Again, the hopper would still end up empty, but the boxcar would not contain all the cargo. An axiomatization of this example can be found in [5].

5.4 Discussion

For the most part, treatments of simultaneous actions in the literature are limited. As mentioned earlier, STRIPS-based systems (*e.g.*, [57; 60; 63]) only allow simultaneity when the actions are independent. A few others (*e.g.*, [47]) allow for synergistic effects cast in terms of domain constraints on states (*e.g.*, in any state s, if the left side of the piano is lifted, and the right side is lifted, then the piano is lifted).

The situation calculus shows more promise with the introduction of action composition operators. For instance, given two actions a_1 and a_2, then the action $a_1 + a_2$ is the action

of performing the two simultaneously (*e.g.*, [22; 52]). Explicit axioms can then be given for these complex actions, and mechanisms can be introduced to automatically derive such axioms from the individual actions if they are independent of each other (*e.g.*, [36]). If the actions are not independent of each other, some reasonable solutions can be found and, as long as actions are instantaneous, it appears that the theory can remain constructive. But these approaches do not seem to be easily extended to handle the more complex cases in which actions have duration and may be temporally related in complex ways. Pelavin [48] contains an extensive analysis of the problems that arise in general with simultaneous interacting actions.

Our approach provides a range of techniques for representing information about interacting actions. In cases where one has detailed knowledge of the causal structure of a domain, the problems can usually be reduced to a level of independent events, and knowledge about interactions are captured by event generation axioms. In other cases, it is more natural to have explicit knowledge about how certain actions interact with events. We believe this flexibility is important for building comprehensive knowledge bases that contain information about a wide range of situations and that are applicable for different reasoning tasks. In addition, we have shown that we can handle these complex cases without introducing any new mechanisms beyond the basic logic discussed in Section 3.

6 Problems and Future Work

One of the most significant open issues in this paper has to do with temporal durations. A realistic characterization of almost all the examples in this paper would require such a capability. For instance, it is not realistic to say that if an agent tries to turn the ignition on the car for any length of time, then the engine will start. If the action is tried for too short a time, the engine probably won't catch. And if the action is tried for too long a time, the starting motor will burn out. So durations play a critical role. At first glance, adding durations does not pose a problem. One simply defines a function that, given an interval, returns a value on some metric scale. A small number of axioms are required to define the appropriate properties of this function. We have not pursued this here as it would further complicate the examples and remove attention from our main points, and in any case, this simple metric model doesn't solve the problem. Consider the action of starting the engine again. There is no minimum time or maximum time within which the engine is guaranteed to start. The duration required depends on the condition of the car, the weather, and many other factors that the agent simply won't have access to. A better formalization of the durational constraints would be that the agent turns the ignition until the engine starts, or a certain time elapses and the agent gives up for fear of burning out the motor, or the battery runs flat. The logic we propose offers no direct solution to this problem, and it remains an important challenge.

Another important issue is the introduction of probabilistic knowledge. By staying within standard first order logic, for example, we are restricted to saying that an event will definitely

occur, or that it might possibly occur. We cannot say that an event is very likely to occur, or that it is very unlikely to occur. Such knowledge is crucial for predicting the likely effects of actions and thus for evaluating the likelihood of success for proposed plans. Further, since all formalisms must make assumptions, the ultimate evaluation should be based on how likely the assumptions are to hold. This is another motivation for favoring the explanation closure technique. It makes the assumptions that are made explicit, and thus potentially available for probabilistic analysis. Some initial work on this is described in [37; 38]. It is much more difficult to see how techniques that build the assumptions into the semantic model could be extended to support probabilistic reasoning.

Finally, it needs to be acknowledged that formalizing knowledge using the more expressive temporal representation can be difficult. Subtle differences in meaning and interactions between axioms may be more common than in less powerful representations, and more experimentation is needed in building knowledge bases based on our representation. But it seems to us that this is the price to pay if we want to move beyond Yale Shooting and the Blocks World. Our representation is based on intuitions about the way people describe and reason about actions in language, which we believe makes it more natural, intuitive, and grounded in common sense.

7 Conclusion

Many of the techniques we have proposed have appeared in various forms previously in the literature. What makes this work novel is the combination of the techniques into a unified, and what we think is an intuitive and relatively simple framework. This logic has the following features:

1. It can express complex temporal relationships because of its underlying temporal logic.
2. It supports explicit reasoning about action attempts and the events they cause, which allows explicit reasoning about success and failure of attempted actions, and subsumes work on conditional effects.
3. It handles external events in a natural way, making only minimal distinctions between external events and events caused by the acting agent.
4. It handles simultaneous actions in a direct manner, including cases of simultaneously interacting actions.

By using an explicit temporal logic with events, we have been able to handle all these problems within a standard first-order logic. We believe this formalism to be more powerful than competing approaches. We also believe that it is simpler than these other formalisms will be once they are extended to handle the temporal complexities of realistic domains, assuming they can be successfully extended.

The second contribution of this paper has been to add to the argument that explicit frame axioms, based on the explanation closure approach, produce a viable theory of prediction with some attractive formal and practical properties. On the formal side, this approach does not require the introduction of specialized syntactic constructs into the language or the use of nonmonotonic semantic theories. On the practical side, the closure axioms capture knowledge that is required in any case as part of the specification of the domain. Note that the formulation of the temporal logic and the use of explanation closure are separable issues. We would not be surprised if the nonmonotonic techniques in the literature could be adapted to our temporal logic. Whether this could be done without extending the syntax with special operators specifically introduced to enable the right minimizations is more questionable, however. The risk is that such operators might subtly affect the range of situations that can be handled, and may require non-intuitive formulations of problems.

While this paper has focussed on formal issues, it is important to remember that our goal is actually the development of practical planning and natural language understanding systems. As a result, another key requirement on this model is that it provides insight and guidance in developing effective knowledge representation systems. We are using this formalism in the TRAINS project [8; 19; 9] and have found that it allows us to express the content of natural language utterances quite directly, and supports plan reasoning algorithms similar to those in the planning literature. The use of explicit assumptions and closure reasoning is essential in such an interactive system where plans are formed based on assumptions and where those assumptions are often the subject of the conversation.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant number IRI-9003841. The Government has certain rights in this material. This work is also supported by ONR/ARPA research grant no. N00014-92-J-1512 and Air Force – Rome Air Development Center research contract no. F30602-91-C-0010.

8 References

[1] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983. Also in *Readings in Knowledge Representation*, R.J. Brachman and H.J. Levesque (eds.), Morgan Kaufmann, 1985, pp. 509–522.

[2] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 464–479.

[3] James F. Allen. Temporal reasoning and planning. In *Reasoning about Plans*, pages 1–68. Morgan Kaufmann, San Mateo, CA, 1991.

[4] James F. Allen. *Natural Language Understanding, 2nd ed.* Benjamin/Cummings Publishing Co., Menlo Park, CA, 1994.

[5] James F. Allen and George Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5):531–579, 1994.

[6] James F. Allen and Patrick J. Hayes. Moments and points in an interval-based temporal logic. *Computational Intelligence*, 5(4):225–238, 1989.

[7] James F. Allen and Johannes A. Koomen. Planning using a temporal world model. In Alan Bundy, editor, *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, pages 741–747, Karlsruhe, West Germany, 8–12 August 1983. William Kaufmann. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 559–565.

[8] James F. Allen and Lenhart K. Schubert. The TRAINS project. TRAINS Technical Note 91-1, Department of Computer Science, University of Rochester, Rochester, NY, 14627, May 1991.

[9] James F. Allen, Lenhart K. Schubert, George Ferguson, Peter Heeman, Chung Hee Hwang, Tsuneaki Kato, Marc Light, Nathaniel G. Martin, Bradford W. Miller, Massimo Poesio, and David R. Traum. The TRAINS project: A case study in defining a conversational planning agent. *Journal of Experimental and Theoretical AI*, 7:7–48, 1995. Also available as TRAINS Technical Note 93-4, Department of Computer Science, University of Rochester.

[10] Fahiem Bacchus, Josh Tenenberg, and Johannes A. Koomen. A non-reified temporal logic. In R.J. Brachman, H.J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 2–10, Toronto, Ont., 15–18 May 1989.

[11] Andrew Baker. Nonmonotonic reasoning in the framework of the situation calculus. *Artificial Intelligence*, 49:5–24, 1991.

[12] Frank M. Brown, editor. *Proceedings of the 1987 Workshop: The Frame Problem in Artificial Intelligence.* Lawrence, KA, Morgan Kaufmann, 12–15 April 1987.

[13] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 537–558.

[14] Donald Davidson. The logical form of action sentences. In N. Rescher, editor, *The Logic of Decision and Action.* University of Pittsburgh Press, 1967. Excerpted in *The Logic of Grammar*, D. Davison and G. Harmon (eds.), Dickenson Publishing Co., 1975, pp. 235–245.

[15] Ernest Davis. Infinite loops in finite time: Some observations. In Bernard Nebel, Charles Rich, and William Swartout, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR92)*, pages 47–58, Boston, MA, 25–29 October 1992. Morgan Kaufmann.

[16] David R. Dowty. The effects of aspectual class on the temporal structure of discourse: Semantics or pragmatics? *Linguistics and Philosophy*, 9(1), 1986.

[17] George Ferguson. Explicit representation of events, actions, and plans for assumption-based plan reasoning. Technical Report 428, Department of Computer Science, University of Rochester, Rochester, NY, June 1992.

[18] George Ferguson and James F. Allen. Generic plan recognition for dialogue systems. In *Proceedings of the ARPA Workshop on Human Language Technology*, Princeton, NJ, 21–23 March 1993.

[19] George Ferguson and James F. Allen. Arguing about plans: Plan representation and reasoning for mixed-initiative planning. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*, pages 43–48, Chicago, IL, 13–15 June 1994.

[20] Richard E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:198–208, 1971. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 88–97.

[21] A. Galton. A critical examination of Allen's theory of action and time. *Artificial Intelligence*, 42(2–3):159–188, 1990.

[22] Michael Gelfond, Vladimir Lifschitz, and Arkady Rabinov. What are the limitations of the situation calculus? In *Proceedings of the AAAI Symposium on Logical Formalizations of Commonsense Reasoning*, pages 59–59. Stanford University, 26–28 March 1991.

[23] Michael P. Georgeff. Actions, processes, and causality. In Michael P. Georgeff and Amy L. Lansky, editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, Los Altos, CA, 30 June–2 July 1986. Morgan Kaufmann.

[24] Michael P. Georgeff. The representation of events in multiagent domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 70–75, Philadelphia, PA, 11–15 August 1986. University of Pennsylvania.

[25] Alvin I. Goldman. *A Theory of Human Action.* Prentice-Hall, Englewood Cliffs, NJ, 1970.

[26] Cordell Green. An application of theorem proving to problem solving. In Donald E. Walker, editor, *Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI-69)*, pages 741–747, Washington, DC, 7–9 May 1969. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 67–87.

[27] Andrew W. Haas. The case for domain-specific frame axioms. In Frank M. Brown, editor, *Proceedings of the 1987 Workshop: The Frame Problem in Artificial Intelligence*, pages 343–348. Lawrence, KA, Morgan Kaufmann, 12–15 April 1987.

[28] C. L. Hamblin. Instants and intervals. In J. T. Fraser, F. C. Haber, and G. H. Müller, editors, *The Study of Time*, pages 324–328. Springer-Verlag, New York, 1972.

[29] Jerry R. Hobbs, Mark E. Stickel, Douglas E. Appelt, and Paul Martin. Interpretation as abduction. *Artificial Intelligence*, 63:69–142, 1993.

[30] Henry A. Kautz. The logic of persistence. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 401–405, Philadelphia, PA, 11–15 August 1986. University of Pennsylvania.

[31] Robert Kowalski. Database updates in the event calculus. *Journal of Logic Programming*, 12:121–146, 1992.

[32] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[33] Peter Ladkin and R. Maddux. Representation and reasoning with convex time intervals. Technical report KES.U.88.2, Kestrel Institution, Palo Alto, CA, 1988.

[34] Vladimir Lifschitz. Formal theories of action. In Frank M. Brown, editor, *Proceedings of the 1987 Workshop: The Frame Problem in Artificial Intelligence*, pages 35–58. Lawrence, KA, Morgan Kaufmann, 12–15 April 1987.

[35] Vladimir Lifschitz and Arkady Rabinov. Miracles in formal theories of action (research note). *Artificial Intelligence*, 38(2):225–238, March 1989.

[36] Fangzhen Lin and Yoav Shoham. Concurrent actions in the situation calculus. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 580–585, San Jose, CA, 12–16 July 1992.

[37] Nathaniel G. Martin. *Using Statistical Inference to Plan Under Uncertainty.* PhD thesis, Department of Computer Science, University of Rochester, Rochester, NY, 1993. To appear as a Technical Report.

[38] Nathaniel G. Martin and James F. Allen. Statistical probabilities for planning. Technical Report 474, Department of Computer Science, University of Rochester, Rochester, NY, November 1993.

[39] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 634–639. MIT Press, 12–19 July 1991.

[40] J. McCarthy. Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1,2):27–39, 1980.

[41] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. American Elsevier Publishing Co., Inc., 1969. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 393–435.

[42] Drew McDermott. Reasoning about plans. In J.R. Hobbs and R.C. Moore, editors, *Formal Theories of the Commonsense World*, pages 269–318. Ablex Publishing, Norwood, NJ, 1985.

[43] Rob Miller and Murray Shanahan. Narratives in the situation calculus. *Journal of Logic and Computation*, 4(5), 1994.

[44] Leora Morgenstern and Lynn A. Stein. Why things go wrong: A formal theory of causal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, St. Paul, MN, 21–26 August 1988. University of Minnesota. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 641–646.

[45] A.P.D. Mourelatos. Events, processes and states. *Linguistics and Philosophy*, 2:415–434, 1978.

[46] Nils J. Nilsson. *Principles of Artificial Intelligence.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1980.

[47] Edwin P.D. Pednault. Formulating multi-agent, dynamic-world problems in the classical planning framework. In M.P. Georgeff and A.L. Lansky, editors, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 47–82, Los Altos, CA, 30 June–2 July 1986. Morgan Kaufmann. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 675–710.

[48] Richard N. Pelavin. Planning with simultaneous actions and external events. In *Reasoning about Plans*, pages 127–212. Morgan Kaufmann, San Mateo, CA, 1991.

[49] Javier Pinto. *Temporal Reasoning in the Situation Calculus.* PhD thesis, University of Toronto, Toronto, Ontario, Canada, February 1994.

[50] Raymond Reiter. The projection problem in the situation calculus: A soundness and completeness result, with an application to database updates. In *Proceedings of the First International Conference on AI Planning Systems*, pages 198–203, College Park, MD, 15–17 June 1992. Morgan Kaufmann.

[51] Erik Sandewall. *Features and Fluents.* Oxford University Press, 1994.

[52] Lenhart Schubert. Monotonic solution of the frame problem in the situation calculus: An efficient method for worlds with fully specified actions. In Henry E. Kyburg, Jr., Ronald P. Loui, and Greg N. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–68. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990. Also available as University of Rochester TR 306, August 1989.

[53] Lenhart K. Schubert and Chung-Hee Hwang. An episodic knowledge representation for narrative text. In R.J. Brachman, H.J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 444–458, Toronto, Ont., 15–18 May 1989. Morgan Kaufmann.

[54] Murray Shanahan. Representing continuous change in the situation calculus. In Luigia Carlucci Aiello, editor, *Proceedings of the European Conference on Artificial Intelligence (ECAI-90)*, pages 598–603, Stockholm, Sweden, 6–10 August 1990. Pitman Publishing Co.

[55] Yoav Shoham. Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence*, 33(1):89–104, 1987.

[56] Yoav Shoham. *Reasoning about change: Time and Causation from the Standpoint of Artificial Intelligence.* MIT Press, Cambridge, MA, 1988.

[57] Austin Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 888–889, Cambridge, MA, 1977. MIT. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 291–296.

[58] Johan F. A. K. van Benthem. *The Logic of Time.* D. Reidel and Kluwer, Dordrecht and Boston, 1983.

[59] Zeno Vendler. *Linguistics in Philosophy.* Cornell University Press, New York, 1967.

[60] Stephen A. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3):246–267, 1983. Also in *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (eds.), Morgan Kaufmann, 1990, pp. 297–318.

[61] Marc Vilain and Henry Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 377–382, Philadelphia, PA, 11–15 August 1986. University of Pennsylvania.

[62] Marc Vilain, Henry Kautz, and Peter van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. In *Readings in Qualitative Reasoning about Physical Systems*, pages 373–381. Morgan Kaufman, San Mateo, CA, 1990.

[63] David E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, San Mateo, CA, 1988.

Appendix E

Performance of Temporal Reasoning Systems*

Abstract

This paper describes the performance evaluation of six temporal reasoning systems. We show that if you are working with large temporal datasets where information is added incrementally throughout the execution of the program, systems using incompletely connected graphs (*i.e.*, TMM, TimeGraph and TimeGraph-II) seem the best option. While they do not offer the constant query time of systems using fully connected graphs (*i.e.* the systems based on constraint satisfaction), the savings at assertion time are so substantial that the relatively small performance penalty for queries is a reasonable tradeoff. Of course, these systems do not offer the expressivity of the interval-based systems as they only handle point-based relations. Of the three, TimeGraph-II offers a wider range of qualitative relations as it handles point inequality. It does not currently handle metric information, however, as do TMM and TimeGraph. Thus decisions between these three may be more determined by the reasoning capabilities required rather than raw performance.

1 Introduction

Research on temporal reasoning has produced many different temporal reasoning systems. These systems differ in expressive power and in computational complexity. While the theoretical complexity of these techniques has been explored in depth (*e.g.*, [11], [8]), there has been no systematic study of their actual performance. This is especially important since it is the behavior of these systems in practice that determine their usefulness in actual implementations. In this paper, we evaluate six different temporal reasoning systems on a range of

*This material is based on: Ed Yampratoom and James F. Allen, "Performance of Temporal Reasoning Systems," *SIGART Bulletin*, 4(3): 26–29, July 1993.

datasets of varying size (up to 9600 intervals). The results suggest that techniques based on full constraint satisfaction (i.e., varying assertion time but constant query time) are unlikely to be useful on large datasets, but systems that use data structures that reduce assertion time but increase query time appear to be effective.

The six systems studied in this report are TimeLogic[6], TimeGraph[9], MATS[4; 5], Tachyon[1], TMM[2; 10], and TimeGraph-II[3]. These systems differ greatly in their underlying mechanisms and expressive power. They can be roughly classified into systems that use constraint satisfaction techniques at assertion time and have a fixed query time (TimeLogic, MATS, and Tachyon), and systems that build graph structures that may require some search at query time to compute an answer (Timegraph, TimeGraph-II, TMM). The latter approach can optimize assertion time, but it is not clear how bad the performance penalty at query time will be. We will give some quantitative characterizations applicable to this question for a specific class of datasets that should commonly arise in planning and scheduling applications.

The systems also vary considerably in their expressive power. TimeLogic and MATS, for instance, support some disjunctive information about time intervals (*e.g.*, interval I is disjoint from interval J, I meets or is during J). TimeGraph and TMM, on the other hand, only support simple point relations ($<$, $=$, $\leq$), and TimeGraph-II supports the full set of point relations ($<$, $=$, $\leq$, $\neq$). Tachyon is essentially a point-based system, but can handle interval disjunctions by returning one possible answer and allow the user to backtrack through the set of possible answers. The systems also differ as to whether quantitative information is handled. TimeLogic and Timegraph-II only handle qualitative constrains, whereas Tachyon, TimeGraph and TMM can handle quantitative constraints. MATS has the most expressive representation and handles both qualitative interval constraints as well as quantitative constraints. All the systems except Tachyon are implemented in Common Lisp, and Tachyon is in C++.

A reasonable performance comparison should be done by looking at how these systems scale up. An absolute comparison is not going to be as useful due to the difference in implementation languages and degree of optimization done on the systems. Since these systems do not have exactly the same expressivity, we should take these differences in power into account in the final judgment.

This report will describe how we generate the dataset to measure these systems, the results of the measurement on *assertion time* and *query time*, and how well each system scales up to the growing amount of data.

2 Temporal Reasoning Systems

As mentioned above, different systems vary greatly in their expressive powers and reasoning mechanisms. In this section, we will look at the details of each system.

TimeLogic[6] is interval-based. It performs transitive closure computations on intervals for qualitative constraints between intervals, using automatically created reference intervals to limit constraint propagations. It cannot handle metric information.

MATS[4; 5] is the most complete system integrating both metric (quantitative) and Allen (qualitative) constraints, including the *disjoint* constraint. Allen constraints are interval-based, while metric constraints are point-based. MATS performs transitive closure computations on intervals for Allen constraints and uses constraint satisfaction for point-based metric reasoning.

Tachyon[1] can handle both metric information and qualitative information. For the *disjoint* constraint, it simplifies the problem by just presenting the first solution found. Since this facility is not comparable to getting the most general solution as provided by other systems, we will not be using this feature. A GUI version of Tachyon allows sequential seek of solutions, however. Tachyon performs constraint satisfaction for point-based metric reasoning. Qualitative constraints are translated into quantitative ones by introducing the concepts of *epsilon* and *infinity*.

TimeGraph[9] can handle both metric and qualitative information but cannot handle *disjoint* relationship. It uses a partial order graph, called timegraph, whose nodes represent time points. Directed links between points indicate relations between the points ($<$ or $\leq$). The timegraph is partitioned into chains, with possible links between chains.

TimeGraph-II[3], the successor of TimeGraph, also uses a timegraph data structure. TimeGraph-II supports the full set of point relations ($<$, $=$, $\leq$, $\neq$). Another improvement of TimeGraph-II over TimeGraph is that it automatically structures the timegraph for efficiency. Since it uses the timegraph data structure, it also cannot handle *disjoint* relationship. The current version of TimeGraph-II cannot handle metric information.

TMM[2; 10] is a temporal database management system. It uses temporal constraint graph (TCG) data structure to construct a time map with simple point relations ($<$, $=$, $\leq$). It can handle both metric and qualitative information but cannot handle *disjoint* relationship.

3 Dataset Generator

To test the systems, we need to generate a large amount of data. We used a dataset generator which, given train schedules, will generate all temporal constraints needed to schedule these trains. We decided not to use randomly generated datasets because large random datasets are rarely consistent. More importantly, it is better to use datasets that are similar to the data that might be used in an actual planning and scheduling applications.

The original datasets are written in KRSL (Knowledge Representation Specification Language)[7], which describes a superset of the statements representable in the six systems. Any temporal information expressible in any of the six systems is thus expressible in KRSL but not necessarily the other way round. Six translators, translating KRSL statements into each

system's input format, automate the task of converting the original datasets into usable formats. These translations are done off-line and the translation time is not taken into account in our evaluation.

Some of the information may not be translatable into a particular system and will be discarded by the translator for that system. For example, all metric information will be ignored by the KRSL-to-TimeLogic translator, and any *disjoint* relations will be ignored by the KRSL-to-Tachyon translator. But since we are currently interested mostly in how these systems scale up, these differences in what the systems can handle will not significantly affect our results.

The dataset generator takes files containing train schedules as input, and writes an output file containing KRSL statements detailing all temporal constraints needed to schedule these trains.

Since we are using a train scheduling domain, we define a language for specifying schedules to simplify the construction. A train schedule must state the timing window in which the train can run, the names of the stations and the tracks it has to go through, and the actions that have to be done at the stations. It also must specify the timing characteristics of the stations and tracks, for example, how long it takes to unload a car in Buffalo, or how fast a train can run on the eastern corridor section between Boston and Providence.

The train schedule input file format is as follows:

```
(earliest-start-time latest-end-time
 number-of-repeats number-of-cars-to-begin-with
   (station1 #-cars-to-load #-cars-to-unload)
   ...
   (stationN #-cars-to-load #-cars-to-unload))

...Timing data of the stations and tracks...
```

The route is circular: the train will loop through the given stations, from *station1* to *stationN* then to *station1* for *number-of-repeats* times.

The timing characteristics of stations and tracks specify the coefficients used in calculating the lower and upper bounds for the time trains spent at stations and travelling on tracks. The values of the bounds are functions of the number of cars and the amount of loading and unloading.

To schedule multiple trains, we use multiple input files. Only one train is allowed to travel on a single track at any given time. So if some trains share the same track, their travelling intervals on that track will be disjoint.

4 Dataset Generator Output

From the information in each input file, the generator will write KRSL statements that will:

- Create intervals corresponding to times spent at stations, time spent travelling between stations, and starting and ending intervals to envelope all other intervals. Example:

```
(create (time-interval) :name 'T5)
```

- Assert a chain of temporal ordering constraints between these intervals, since the whole set is fully ordered for each train. Example:

```
(assert (interval-before T0 T1))
(assert (interval-before-bounds T0 T1
  #?(create (duration-bounds :zero :zero))))
```

- Assert absolute time constraints on starting and ending intervals: the starting interval cannot begin before *earliest-start-time* and the ending interval cannot end after *latest-end-time*. Example:

```
(assert (pt>= (interval-start T0) 20))
(assert (pt<= (interval-end T5) 500))
```

- Assert absolute-time durational constraints on the other intervals, using stations' and tracks' timing characteristics to calculate the duration bounds of each interval. Example:

```
(assert (interval-bounds T1
  #?(create (duration-bounds
      (:minutes 23) (:minutes 27)))))
```

- Assert *disjoint* constraints when scheduling multiple trains. We need to declare all intervals using the same track to be disjoint because only one train can use that track at one time. Example:

```
(assert (interval-disjoint T5 T7))
```

5 Measurements

The performance test was done on a Sun SparcStation 10/30 with 64 megabytes of physical memory. The five Common Lisp systems were tested using their compiled code on Allegro CL 4.1, while the test on Tachyon used an executable image supplied by its authors.

Since these temporal systems differ in the ways they make use of the given data, a straightforward running time comparison is not possible. To generalize, temporal reasoning systems spend time in three phases:

- Load the data from files and convert them into appropriate data structures.
- Perform intermediate calculation steps.

- Retrieve and/or compute the answers to queries.

From the user's point of view, there are two distinct phases: *assertion time* and *query time*. The time spent in the last phase is querying time. Should we consider only the first phase as assertion time or must we include the second phase as well?

MATS, TMM, and TimeGraph-II separate the first two phases, while TimeGraph and TimeLogic interleave them in such a way that it is not possible to measure them separately. Therefore, we consider assertion time of these systems to be the toal of the first two phases. Tachyon doesn't accept queries at all. Instead, after the input file is read, Tachyon outputs all intervals' ranges of start time, end time, and duration. Its time is thus measured from the beginning to the end of its execution, when it prints out information for all intervals.

The query time of MATS and TimeLogic are constant since they only need to look up the answers from tables. The query time of TMM, TimeGraph, and TimeGraph-II are not constant. We will compare query time performance in section 7.

In summary, the time measured is *assertion time* plus any additional computation time needed before a query could be performed. Moreover, the assertion is done on the subset of the original dataset that each system can handle. Since we are not going to focus on absolute performance, but rather on how well each system scale up, the slight differences between datasets should not matter.

6 Assertion Time Measurement

The performance comparison of the six systems is shown in Figure 1. We don't have data for MATS and TimeLogic after 2000 statements due to both systems requiring too much memory. Note how the performance plot divides the systems into two groups.

On small datasets (less than 1000 KRSL statements), the six systems are about equal in scalability (except for MATS), as can be seen in Figure 2. The performances only differ by a constant while the slopes are roughly similar. But MATS and TimeLogic then explode around 1000 KRSL statements in Figure 1.

On large datasets, the systems can be further divided into two classes as can be seen in Figure 3. Note that even though Tachyon is written in C++, the assumed speed advantage is lost when the datasets become large.

7 Query Time Measurement

Since TMM and TimeGraph-II do not have constant query time, it is important to test how these systems behave on large datasets. That way, we may be able to better quantify the tradeoff between the advantage these systems have at assertion time and the penalty they pay at query time.

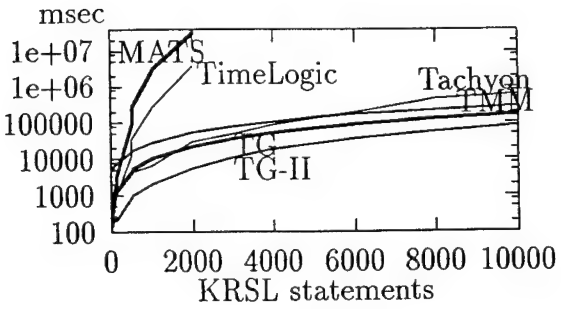


Figure 1: Six systems compared (log scale)

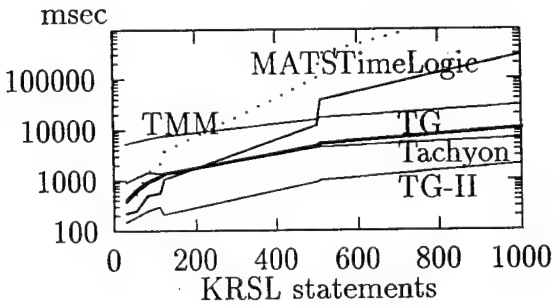


Figure 2: Performance on small datasets (log scale)

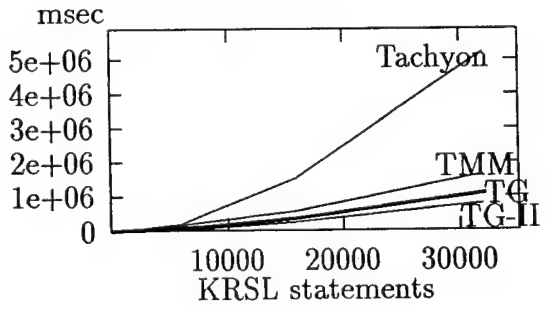


Figure 3: Performance on large datasets (linear scale)

size (KRSL stmts)	no. of trains	TMM av. query time	TG-II av. query time
1000	5	73.266	4.501
2000	10	74.600	5.246
3000	15	73.607	5.115
4000	20	75.717	5.536
5000	25	73.469	5.232
10000	50	72.146	5.133

Table 1: Query time doesn't grow with data size

cross-chain links	TMM av. query time	TG-II av. query time
0	67.537	5.385
50	64.745	1.519
100	67.779	1.396
250	68.225	1.577
500	68.050	1.500
1000	68.735	1.519
1500	66.993	1.365

Table 2: Query time doesn't grow with cross-chain links

To test query behavior, we create a set of train schedules with increasing number of trains, from 5 to 50 trains. To prevent the systems from taking an advantage of the regular nature of the datasets, the datasets are scrambled so that the chain nature is not obvious. Moreover, the metric information is discarded so that the comparison is more just.

Table 1 shows the average TMM and TimeGraph-II query time for the dataset. Each average comes from performing 1000 random qualitative queries. The unit of time is millisecond.

As can be seen from the tables, the query time doesn't show an increasing trend when the dataset size grows.

To test how an increasing number of cross-chain links affect query time, we create another dataset containing 51 chains with 10200 KRSL statements. We then measure random query time on this dataset, adding more cross-chain links for each round of measurement while keeping the dataset consistent.

Table 2 shows the average TMM and TimeGraph-II query time when the number of cross-chain links increases. Each average comes from performing 1000 random qualitative queries. The unit of time is millisecond. The maximum query time for TimeGraph-II is 50 msec, for TMM is 1800 msec.

For the constant-query time systems, the average MATS random query time is 7.862 msec, while that of TimeLogic is 0.649 msec. Both systems are measured on a much smaller

dataset (500 KRSL statements, with 138 intervals) since they cannot handle large dataset as mentioned in section 6. Tachyon query time cannot be measured since it just prints all information for all intervals at the end of its run.

8 Conclusions

It is clear from the results that if you are working with large temporal datasets where information is added incrementally throughout the execution of the program, systems using incompletely connected graphs (*i.e.*, TMM, TimeGraph and TimeGraph-II) seem the best option. While they do not offer the constant query time of systems using fully connected graphs (*i.e.* systems based on constraint satisfaction), the savings at assertion time are so substantial that the relatively small performance penalty for queries is a reasonable tradeoff. Of course, these systems do not offer the expressivity of the interval-based systems as they only handle point-based relations. Of the three, TimeGraph-II offers a wider range of qualitative relations as it handles point inequality. It does not currently handle metric information, however, as do TMM and TimeGraph. Thus decisions between these three may be more determined by the reasoning capabilities required rather than raw performance.

If assertion time is not an issue, say in an application where the database could be constructed once in advance and then only queried, then the constraint satisfaction approaches could be used. There will still be a significant memory requirement for the final database, however, as these systems all used fully connected graphs. In addition, the performance of TimeLogic and MATS is so bad on large datasets that it still might not be practical to even compute the database in advance! It may be, however, that these two systems are paying unnecessary penalty due to garbage collection, and that a new algorithm could be developed that does not create so many temporary objects.

With small datasets, all of these systems may be used effectively and the choice may be more determined by the expressivity and other reasoning facilities provided by each system. For instance, the interval-based relations have been useful in developing planning algorithms and in natural language applications. Thus, as long as the datasets remain small, a researcher can effectively use systems such as TimeLogic and MATS. As another example, TMM offers facilities for reasoning about persistence of facts that are not provided in the other systems. Since this type of reasoning is critical in many applications, TMM would be a good choice. If execution time is still critical, however, then the tradeoff must still be considered between assertion and query time. If query time is of paramount importance, then the constant query-time algorithms might be selected, although the measured average query time shows that the penalty incurred by non-constant query time algorithms is negligible in most cases. And if the applications require that we always keep one consistent scenario, then Tachyon's approach of providing a solution, instead of the most general solution, is more appropriate.

There are some qualifications that must be attached to our conclusions, however. The datasets we have used have considerable structure. In particular, the data is predominantly

organized into chains of times. We believe that naturally occuring data in applications such as planning and scheduling, and natural language, will tend to have this structure. While the structure has no effect on the performance of the systems based on constraint satisfaction, it does have significant effect on TMM and the TimeGraph systems. In particular, it is the structure that allows these systems to have reasonable query times and thus be viable. As a result, we would expect TimeGraph and TimeGraph-II (and possibly TMM), to behave quite differently on randomly generated data. But since we are interested in evaluating these systems for use in applications, their performance on random data may have little or no practical significance.

9 Acknowledgements

This material is based upon work supported in part by U.S. Air Force – Rome Laboratory research contract no. F30602-91-C-0010. We also would like to thank all the authors of the systems whose works we evaluate here.

10 References

[1] R. Arthur and J. Stillman, *Temporal Reasoning for Planning and Scheduling*, Artificial Intelligence Laboratory, General Electric Research and Development Center, 1992.

[2] T. Dean and D. McDermott, "Temporal data base management," *Artificial Intelligence*, 32:1–55, 1987.

[3] A. Gerevini and L. K. Schubert, "Efficient temporal reasoning through timegraphs," To appear in Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, 1993.

[4] H. Kautz, *MATS (Metric/Allen Time System) Documentation*, AT&T Bell Laboratories, 1991.

[5] H. Kautz and P. Ladkin, "Integrating Metric and Qualitative Temporal Reasoning," In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, 12–19 July 1991.

[6] J. Koomen, "The TIMELOGIC Temporal Reasoning System," Technical Report 231, Computer Science Department, University of Rochester, Rochester, NY, March 1989, Revised.

[7] N. Lehrer, *et al.*, "Knowledge Representation Specification Language (KRSL) 2.0," manuscript, 1992.

[8] I. Meiri, "Combining qualitative and quantitative constraints in temporalreasoning," In *Proceedings of the Ninth National Conference on Artificial Intlligence (AAAI-91)*, pages 260–267, 12–19 July 1991.

[9] S. Miller and L. K. Schubert, "Time revisited," *Computational Intelligence*, 6:108–118, 1990.

[10] R. Schrag, J. Carciofini, and M. Boddy, "Beta-TMM Manual (version b19)," Technical Report CS-R92-012, Honeywell SRC, 1992.

[11] M. Vilain, H. Kautz, and P. van Beek, "Constraint propagation algorithms for temporal reasoning: a revised report," In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems.* Morgan Kaufmann, 1990.

Appendix F

On Computing the Minimal Labels in Time Point Algebra Networks*

Abstract

We analyze the problem of computing the minimal labels for a network of temporal relations in the Point Algebra. van Beek proposes an algorithm for accomplishing this task which takes $O(max(n^3, n^2 \cdot m))$ time (for n points and m $\neq$-relations). We show that the proof of the correctness of this algorithm given by van Beek and Cohen is faulty, and we provide a new proof showing that the algorithm is indeed correct.

1 Introduction

The *Interval Algebra* (IA) (Allen 1983) and the *Point Algebra* (PA) (Vilain and Kautz 1986) are two fundamental approaches to representing temporal information in terms of qualitative relations between intervals and points respectively. The Interval algebra is based on thirteeen basic relations that can hold between intervals; from these, 2^{13} possible disjunctive relations can be derived. The elements of the Point Algebra are the relations $\{\emptyset, <, \leq, =, >, \geq, \neq, ?\}$, where ? is the disjunction of the three basic relations $<$,$=$ and $>$. The "pointizable" interval algebra (written as SIA in (van Beek 1990)) is a restricted algebra of IA which consists of the set of the 188 relations in IA (including the empty relation) that can be translated into conjunctions of point relations between the endpoints of the intervals (Ladkin and Maddux 1988; van Beek and Cohen 1990).

A set of asserted relations in IA (PA) can be represented through a labeled graph which van Beek (1992) calls IA-network (PA-network), whose vertices represents interval variables

*This material is based on: Alfonso Gerevini and Lenhart Schubert, "On Computing the Minimal Labels in Time Point Algebra Networks," *Computational Intelligence*, 11(3): 443–448, 1995.

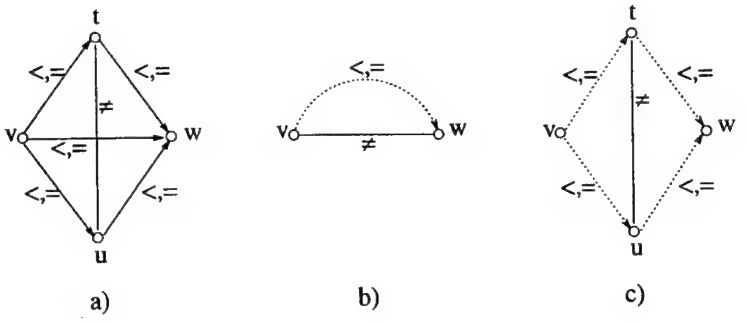


Figure 1: a) van Beek's forbidden graph; b) and c) the two kinds of implicit $<$ relation. Dotted arrows indicate paths, solid lines $\neq$-edges. In all the graphs there is an implicit $<$ relation between v and w.

(point variables) and whose labeled edges correspond to the asserted relations. Given a set of asserted temporal relations (a labeled graph), one of the main reasoning tasks concerns deducing new relations from those that are known and, in particular, computing the strongest entailed relations for each pair of intervals or points (called the "minimal labels" in (van Beek and Cohen 1990), the "feasible relations" in (van Beek 1992), and the "closure" of the assertions in (Vilain *et al.* 1990)).

In (Vilain and Kautz 1986) an $O(n^3)$ time algorithm for determining the closure of a set of assertions of relations in PA (for n points) is given. In (van Beek 1990; van Beek and Cohen 1990; Vilain *et al.* 1990) it is shown that this algorithm is not correct and in (van Beek 1990; van Beek 1992) another algorithm for accomplishing the same task is provided. The new algorithm takes $O(max(n^3, n^2 \cdot m))$ time and $O(n^2)$ space (for m $\neq$-assertions). Given a PA-network the algorithm performs two main steps: first it computes the *path-consistent network* (Montanari 1974; Mackworth 1977) of the initial network in $O(n^3)$ time; secondly it computes the minimal labels by finding and removing the *forbidden subgraphs* of the path-consistent network (see Figure 1.a) which is accomplished in $O(n^2 \cdot m)$ time. The correctness of this algorithm relies on a lemma given by van Beek and Cohen (1990) claiming that any path-consistent PA-network which does not have minimal labels contains a forbidden graph.

In this note we show that the proof of this lemma is not correct, and we provide a new proof showing that the algorithm given by van Beek (1990) is indeed correct.

2 Representing temporal relations through TL-graphs

In this section we first introduce the necessary terminology and theoretical background (partly taken from (Gerevini and Schubert 1993a)).

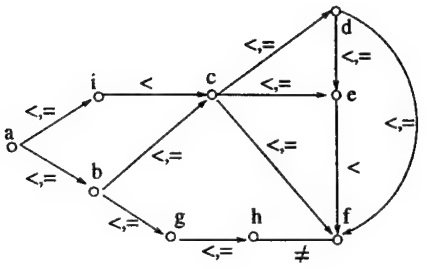


Figure 2: An example of TL-graph

Definition 1 A *temporally labeled graph* (*TL-graph*) is a graph with at least one vertex and a set of labeled edges, where each edge (v, l, w) connects a pair of distinct vertices v, w. The edges are either directed and labeled $\leq$ or $<$, or undirected and labeled $\neq$.

Figure 2 shows an example of TL-graph. We assume that every vertex of a TL-graph has at least one name (time point variable) attached to it. If a vertex has more than one name, then they are alternative names for the same time point. The name sets of any two vertices are required to be disjoint.

More formally, writing P for the set of time-point variables attached to a TL-graph, and V for its set of vertices, we assume that there exists a surjective function PV from P to V.

A *model* of a TL-graph is an interpretation of the vertex names as elements of a totally ordered set T (with strict ordering $<$), such that all names attached to the same vertex denote the same element and the interpretations of names attached to distinct vertices satisfy the constraints expressed by the edge(s), if any, connecting those vertices.

More formally, the notions of an interpretation and of a model for a TL-graph can be specified as follows:

Definition 2 Given a TL-graph G, an *interpretation* of G is a triple $\langle T, I, R\rangle$, where T is a totally ordered set (with ordering $<$); I is a function $I : P \rightarrow T$ such that for all $p_1, p_2 \in$ P *if* $PV(p_1)$=$PV(p_2)$ *then* $I(p_1) = I(p_2)$; R is a function mapping each label l on the edges of G ("$<$","$\leq$" or "$\neq$") into the corresponding binary relation $R(l)$ on T ($<$, $\leq$ or $\neq$).[1]

Definition 3 Given a TL-graph G, a *model* of G is an interpretation such that if (v_1, l, v_2) is an edge of G, then for all $p_i, p_j \in$ P satisfying $PV(p_i) = v_1$ and $PV(p_j) = v_2$

$$< I(p_i), I(p_j) > \in R(l).$$

[1]For instance if $l =$"$<$" then $R(<)$ is the $<$-relation, i.e., the set of pairs $\langle t_1, t_2\rangle$ such that $t_1 < t_2$ and $t_1, t_2 \in T$. (Analogously for "$\leq$" and for "$\neq$".)

Definition 4 A TL-graph is *consistent* if and only if it has at least one model.

Definition 5 Two or more TL-graphs are *logically equivalent* if and only if they have the same models.

A path of length n from v_0 to v_n is a sequence of n triples (v_0, l_1, v_1),..., (v_{n-1}, l_n, v_n) where v_i $(0 \leq i \leq n)$ are vertices and l_j $(1 \leq j \leq n)$ are labels (relations) on directed edges.

Definition 6 In a TL-graph a path is a *$\leq$-path* if each label l_j is $\leq$ or $<$. A $\leq$-path is a *$<$-path* if at least one of these labels is $<$.

Definition 7 A $\leq$-path ($<$-path) of length n from v_0 to v_n, $n \geq 1$, is a *$\leq$-cycle* (*$<$-cycle*) if $v_0 = v_n$. A TL-graph is *acyclic* if it does not contain any $\leq$-cycle.

In (Gerevini and Schubert 1993a) we proved the following theorems about determining consistency of a TL-graph [2]:

Theorem 1 *A TL-graph is consistent iff it does not contain any $<$-cycle, or any $\leq$-cycle that has two vertices connected by an edge with label $\neq$.*

Theorem 2 *A TL-graph can be recognized as being inconsistent, or if it is consistent, collapsed into a logically equivalent acyclic TL-graph in $O(e)$ time, where e is the number of edges.*

Definition 8 A TL-graph contains an *implicit* $<$ relation $v < w$ if there is no $<$-path from v to w and either there is an edge between v and w with label $\neq$ and a $\leq$-path (but no $<$-path) from v to w (see Figure 1.b)); or there exist two vertices u and t such that there is an edge between u and t with label $\neq$ and $\leq$-paths (but no $<$-path) from v to u, u to w, v to t, and t to w (see Figure 1.c)). The graphs isomorphic to the one given in Figure 1.c) are called *$\neq$-diamonds*.

Definition 9 An *explicit graph* for a given TL-graph G is an acyclic TL-graph logically equivalent to G and with no implicit $<$ relations.

Theorem 3 (Gerevini and Schubert 1993a) *An explicit TL-graph entails $v \leq w$ iff there is a $\leq$-path from v to w; it entails $v < w$ iff there is a $<$-path from v to w, and it entails $v \neq w$ iff there is a $<$-path from v to w or from w to v, or there is an edge $(v, \neq, w)$.*

[2] Results similar to Theorems 1 and 2 are also given by van Beek (1990; 1992).

3 Computing the minimal labels

In this section we first introduce the notions of a path-consistent TL-graph and of a minimal TL-graph. We then provide a new proof of van Beek and Cohen's Lemma about forbidden graphs in path-consistent PA-neworks. Given a PA-network $\mathcal{T}$, this lemma allows the computation of the minimal labels of $\mathcal{T}$ by identifying all the forbidden graphs in the path-consistent network of $\mathcal{T}$ and by making explicit all the implicit $<$ relations.

Definition 10 A TL-graph is *path-consistent* if there is exactly one edge joining each pair of vertices, and for every triple (u, v, w) of vertices, any of the possible relations $\{<, >, =\}$ allowed by the label of the edge joining u and v (l') is consistent with the relations allowed by the labels of the edges joining u and w (l''), and v and w (l'''). That is, there exists an interpretation I such that for all p_i, p_j, p_k satisfying $PV(p_i) = u$, $PV(p_j) = v$ and $PV(p_k) = w$

$$< I(p_i), I(p_j) > \in R(l')$$
$$< I(p_i), I(p_k) > \in R(l'')$$
$$< I(p_j), I(p_k) > \in R(l''').$$

Definition 11 The *minimal graph* of a consistent TL-graph G is a TL-graph logically equivalent to G, containing exactly one labeled edge for each pair of distinct vertices and in which for each pair (v_1, v_2) of vertices the label l of the edge (v_1, l, v_2) corresponds to the strongest relation entailed by G [3]. The minimal graph of an inconsistent TL-graph is the empty graph.

The definitions of a path-consistent TL-graph and of a minimal TL-graph are a slight departure from the notions of a *path-consistent network* and of a *minimal network* originally given in the context of constraint satisfaction problems (Montanari 1974; Mackworth 1977). Note also that not every *PA-network* as defined by van Beek and Cohen (1990) is a TL-graph; only networks without $=$-edges translate directly into TL-graphs.

An explicit TL-graph can be transformed into a path-consistent graph by three operations: unifying multiple edges between two vertices into a single edge labeled with the intersection of the labels on the individual edges; introducing new edges in order to have exactly one edge joining each pair of vertices whose label is chosen according to path-consistency and reducing the labels on the original edges in order to satisfy path-consistency.

van Beek and Cohen (1990) give a lemma stating that any path-consistent nonmininmal PA-network of relations taken from PA must include a four-vertex subgraph isomorphic to the graph in Figure 1.a). The proof they provide is not correct but the Lemma is indeed valid. The incorrectness of their proof becomes evident if one observes that the PA-network of figure 1.a) is not the only four-vertex *path-consistent PA-network* that, up to isomorphism, can be derived in the last step of the proof of van Beek and Cohen. In fact, the network

[3] Relation vRw is stronger than relation $vR'w$ if and only if vRw entails $vR'w$, but not the converse; in other words, viewed as subsets of $\{<, =, >\}$, R is a proper subset of R'.

obtained by replacing $(v, \{<, =\}, u)$ with $(v, ?, u)$ and $(u, \{<, =\}, w)$ with $(u, ?, w)$ is another possible network. This contradicts what van Beek and Cohen assert at the end of their proof.

The next theorem provides a new proof of van Beek and Cohen's Lemma. In the first part of the proof, we deal with networks without =-edges, which therefore amount to TL-graphs. In the second part of the proof we relate networks with =-edges to TL-graphs, by talking about the networks obtained by collapsing equal vertices. For these collapsed networks, the first part of the proof applies, and we can infer the existence of a forbidden subgraph. We then argue that the forbidden subgraph must also have been present in the network prior to collapsing it.

Theorem 4 *Any path-consistent PA-network G that contains no forbidden graphs is minimal.*

Proof. Consider the case where G contains no =-edges. In this case it suffices to show that such a network G is already explicit, and that an explicit, path-consistent network is minimal. To see that G is already explicit, note first that there are no implicit <-relations of the first kind in G, since the presence of an edge $(v, \neq, w)$ across a $\leq$-path from v to w would violate path consistency (the edge from v to w would have to be labeled $<$ rather than $\neq$). Moreover, there are no implicit <-relations of the second kind in G, since these can exist only when there is a $\neq$-diamond, and in a path-consistent graph the presence of such a diamond entails the presence of a forbidden subgraph (i.e., each of the four $\leq$-paths making up the $\neq$-diamond must join two vertices whose connecting edge is also labeled $\leq$; the label cannot be stronger ($<$) since otherwise the definition of a $\neq$-diamond is not satisfied). Also there are no $\leq$-cycles in a path-consistent network without =-edges, and so G is indeed explicit. But an explicit, path-consistent network is minimal, since by Theorem 3 the graph entails (a) $v \leq w$ *iff* there is a $\leq$-path from v to w, (b) $v < w$ *iff* there is a <-path from v to w, and (c) $v \neq w$ *iff* there is a <-path from v to w or from w to v or a $\neq$-edge connecting v and w, and by path consistency these three cases respectively entail that the label l in edge (v, l, w) is (a) $\leq$ only if the graph does not entail $v < w$, (b) $<$, and (c) $<$, $>$, or $\neq$.

This leaves us with the case where G contains =-edges (but is still path-consistent and contains no forbidden subgraphs). In this case G will clearly be minimal as long as for any v, w connected by a =-path (a path with label "=" on all edges) the label of the edge connecting v and w is =, and the graph G$'$ obtained by collapsing equated sets of vertices into a single vertex is minimal. The former is true by path consistency, and the latter is true by the first part of the proof (collapsing equated vertices does not destroy path consistency, does not create new forbidden graphs and preserves entailments). □

4 Conclusions

In this note we have provided a new proof of van Beek and Cohen's lemma about some strict orderings that cannot be derived by computing path consistency in a network of temporal

relations in the Point Algebra. This lemma is fundamental in showing that the strongest entailed relations for each pair of points (the minimal labels) of a set of assertions in the Point Algebra can be computed in polynomial time ($O(max(n^2 \cdot m, n^3))$, where n is the number of time points and m is the number $\neq$ assertions (van Beek 1990).

The larger goal which prompted this work was the design of a temporal reasoning system called *TimeGraph-II* (TG-II) (Gerevini and Schubert 1993a; Gerevini and Schubert 1993b; Gerevini and Schubert 1993c). TG-II is descended from TG-I, an earlier system to support natural language understanding (Schubert *et al.* 1983; Schubert *et al.* 1987; Miller and Schubert 1990). TG-II is aimed at a broader class of applications, including planning.

5 References

Allen, J. F. 1993. Maintaining knowledge about temporal intervals. *Communication of the ACM*, 26(1):832–843.

Gerevini, A., and L. K. Schubert 1993a. Efficient temporal reasoning through timegraphs. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambéry, Savoie, France.

Gerevini, A., and L. K. Schubert 1993b. Temporal reasoning in Timegraph I-II. *SIGART Bulletin*, 4(3), July 1993.

Gerevini, A., and L. K. Schubert 1993c. Efficient algorithms for qualitative reasoning about time. To appear in *Artificial Intelligence.*

Ladkin, P., and R. Maddux. 1988. On binary constraint problems. Tecnical Report KES.U.88.8, Kestrel Institute, Palo Alto, CA.

Mackworth, A. K. 1977. Consistency in network of relations. *Artificial Intelligence*, 8(1):99–118.

Miller, S. A., and L. K. Schubert. 1990. Time revisited. *Computational Intelligence*, 6:108–118.

Montanari, U. 1974. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7(3):95–132.

Schubert, L. K., M. A. Papalaskaris, and J. Taugher. 1983. Determining type, part, colour, and time relationships. *Computer (special issue on Knowledge Representation)*, 16:53–60.

Schubert, L. K., M. A. Papalaskaris. and J. Taugher. 1987. Accelerating deductive inference: Special methods for taxonomies, colours, and times. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 187–220. Springer-Verlag, Berlin, Heidelberg, New York.

van Beek, P. 1990. Reasoning about qualitative temporal information. In *Proceedings of the Eighth National Conference of the American Association for Artificial Intelligence*, pages 728–734, Boston.

van Beek, P. 1992 Reasoning about qualitative temporal information. *Artificial Intelligence*, 58(1-3):297–321.

van Beek, P., and R. Cohen. 1990. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–144.

Vilain, M., and H. Kautz. 1986. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the Fifth National Conference of the American Association for Artificial Intelligence*, pages 377–382, Philadelphia, PA.

Vilain M., H. Kautz, and P. van Beek. 1990. Constraint propagation algorithms for temporal reasoning: a revised report. In *Readings in Qualitative Reasoning about Physical Systems*, pages 373–381. Morgan Kaufman, San Mateo, CA.

Appendix G

The Temporal Reasoning Tools TimeGraph I-II*

Abstract

We describe two domain-independent temporal reasoning systems called TimeGraph I and II which can be used in AI applications as tools for efficiently managing large sets of relations in the Point Algebra, in the Interval Algebra, and metric information such as absolute times and durations. Our representation of time is based on *timegraphs*, graphs partitioned into a set of chains on which the search is supported by a metagraph data structure. TimeGraph I was originally developed by Taugher, Schubert and Miller in the context of story comprehension. TimeGraph II provides useful extensions, including efficient algorithms for handing inequations, and relations expressing point-interval exclusion and interval disjointness. These extensions make the system much more expressive in the representation of qualitative information and suitable for a large class of applications.

1 Introduction

We are interested in developing temporal reasoning tools for efficiently managing large data sets of temporal information. Our approach was originally developed by Schubert, Taugher and Schaeffer (formerly Miller) in the context of natural language comprehension [28, 29, 22], and then extended with the aim of making it suitable for a larger class of Artificial Intelligence applications including, in particular, planning [10, 11].

In natural language applications such as story understanding, reasoning about the temporal structure of *episodes* is an important task which requires specialized temporal inference

*This material is based on: Alfonso Gerevini, Lenhart Schubert, and Stephanie Schaeffer, "The Temporal Reasoning Tools TimeGraph I-II", *International Journal of Artificial Intelligence Tools*, 4(1-2): 281–299, 1995.

methods [22]. For example, to answer a question like "Was the wolf alive at any time after everyone (the human characters in the story of Little Red Riding Hood) talked or after everyone ate the goodies?" requires detection of an an inconsistency between a presumed "alive" episode and a semi-infinite "not alive" episode that starts with the death of wolf at the hands of the woodcutter, an event known to precede both the "talking" and "goodie eating" episodes [22].

In planning, constraint-based approaches like Allen and Koomen's [4, 3] and nonlinear planners like UCPOP rely on a temporal reasoning module for efficiency [25, 36]. In fact, the first requires an efficient method for managing relations in the *Interval Algebra* (IA) [2] formed from subsets of thirteen basic relation (see Figure 1); the second requires a specialized reasoning module to maintain the consistency of a temporal database of point relations expressing strict ordering and point-interval exclusion, i.e., relations of the form "$p_1 < p_2$", and "$p_3 < p_4$ or $p_5 < p_3$", where $t_4 < t_5$.[1]

In this paper we present two domain-independent temporal reasoning systems called *TimeGraph I* (TG-I) and *TimeGraph II* (TG-II). These systems can efficiently handle point relations in the Point Algebra (PA) [35, 32, 13], the interval relations in the subclass of IA called SIA [31], and metric information such as constraints involving absolute times and durations. Moreover, a recent extension of TG-II [11] allows the system to efficiently manage point-interval exclusion, as well as the interval disjointness relation "before or after", which is important in planning and scheduling applications.[2]

Temporal relations are represented through graphs, called *timegraphs*, whose vertices represent time points and whose edges represent temporal relations. The main characteristics of timegraphs are their partitioning into a set of chains, which are sets of linearly ordered points, and their use of a *metagraph* structure to guide the search processes across the chains [12].

Given a set of temporal relations S, the main temporal reasoning tasks addressed concern determining the consistency of S and providing the strongest relation entailed by S between (any) two time points. Furthermore, our graphical representation is also suitable for the task of finding *consistent scenarios* (i.e. interpretations for all the temporal variables involved); as shown in [31], this can be efficiently accomplished by performing a topological sort on the timegraph.

Space requirements are one of the major limitations of temporal reasoning systems based on constraint propagation such as Timelogic [18] and MATS [17]. One advantage of our method with respect to the constraint propagation approach (such as [35, 33, 8, 20]) is that the space complexity can be linear in the number of stipulated relations instead of

[1]The "$<$" relation is used to partially order the actions of the plan; the point-interval exclusion relation is used to "protect" action preconditions during planning, where an earlier action may serve to achieve the preconditions of a later one, and no further action should be inserted between them which would subvert those preconditions [36].

[2]For example, planned actions often cannot be scheduled concurrently because they contend for the same resources (agents, vehicles, tools, pathways, etc.).

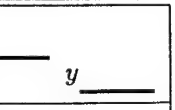
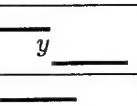
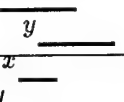
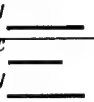
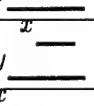
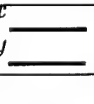
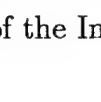
Relation	Symbol	Meaning
x before y y after x	B A	
x meets y y met-by x	M M-	
x overlaps y y overlapped-by x	O O-	
x during y y contains x	D D-	
x starts y y started-by x	S S-	
x finishes y y finished-by x	F F-	
x equal y	E	

Figure 1: The thirteen basic relations of the Interval Algebra

quadratic in the number of time points. The other advantage is the efficiency in terms of computation time in domains such as planning [1] and story understanding in which the temporal data tend to fall naturally into chain-like aggregates [30, 22]. Building timegraphs in practice takes much less time than computing closure (the *minimal network* in constraint propagation terminology [23]), and the amount of time spent in querying relations is nearly constant.

Section 2 provides the basic definitions and terminology for representing temporal relations through timegraphs from [29, 22, 10, 12]. Section 3 briefly shows how a timegraph is constructed from a given set of relations. Sections 4 and 5 describe algorithms for querying a timegraph and managing disjunctions. Section 6 presents the TG-I and TG-II systems. Section 7 reports some experimental results from TG-II. Section 8 gives conclusions and future work.

2 Representing temporal relations through Timegraphs

A *temporally labeled graph* (*TL-graph*) is a graph with at least one vertex and a set of labeled edges, where each edge (v, l, w) connects a pair of distinct vertices v, w. The edges are either directed and labeled $\leq$ or $<$, or undirected and labeled $\neq$.
We assume that every vertex of a TL-graph has at least one name attached to it. If a vertex has more than one name, then they are alternative names for the same time point. The name sets of any two vertices are required to be disjoint. Figure 3 shows an example of

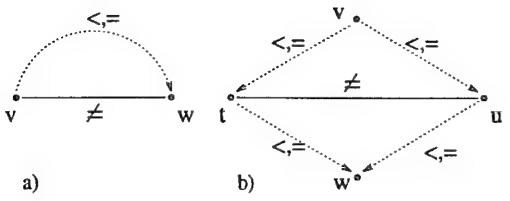


Figure 2: The two kinds of implicit $<$ relation. Dotted arrows indicate paths, solid lines $\neq$ edges. In both graphs there is an implicit $<$ relation between v and w.

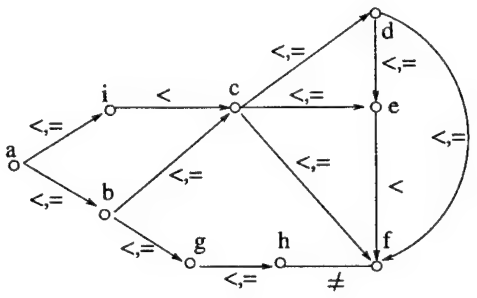


Figure 3: An example of TL-graph.

TL-graph. A path in a TL-graph is called a $\leq$-path if each edge on the path has label $<$ or $\leq$. A $\leq$-path is called a $<$-path if at least one of the edges has label $<$.

We say that a TL-graph G contains an *implicit* $<$ relation between two vertices v_1, v_2 when the strongest relation entailed by the set of relations from which G has been built is $v_1 < v_2$ and there is no $<$-path from v_1 to v_2 in G. Figure 2 shows the two possible TL-graphs which give rise to an implicit $<$ relation. All TL-graphs with an implicit $<$ relation contain one of these graphs as subgraph.

A TL-graph without implicit $<$ relations is an *explicit* TL-graph. In order to make explicit a TL-graph containing implicit $<$ relations we can add new edges with label $<$ (for example in Figure 2 we add to the graph the edge $(v, <, w)$).

An important property of an explicit TL-graph is that it entails $v \leq w$ if and only if there is a $\leq$-path from v to w; it entails $v < w$ if and only if there is a $<$-path from v to w, and it entails $v \neq w$ if and only if there is a $<$-path from v to w or from w to v, or there is an edge $(v, \neq, w)$.

A *timegraph* is an acyclic TL-graph partitioned into a set of *time chains*, such that each vertex is on one and only one time chain. A time chain is a $\leq$-path, plus possibly *transitive edges* connecting pairs of vertices on the $\leq$-path. Distinct chains of a timegraph can be connected by *cross-edges*. Vertices connected by cross-edges are called *metavertices*. Cross-edges and certain auxiliary edges connecting metavertices on the same chain are called

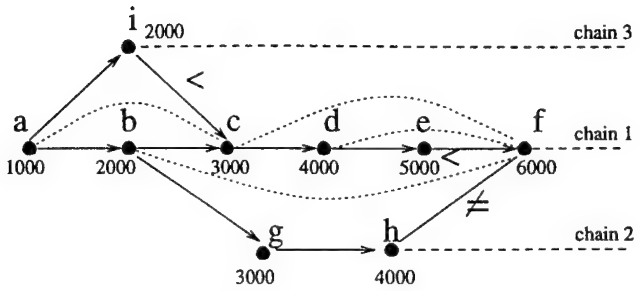


Figure 4: Timegraph of the TL-graph of Figure 2, with transitive edges and auxiliary edges omitted.

metaedges. The metavertices and metaedges of a timegraph T form the *metagraph* of T.

Figure 4 shows the timegraph built from the TL-graph of Figure 3. All vertices except d and e are metavertices. The edges connecting vertices a to i, i to c, b to g, h with f, are metaedges. Dotted edges are special links called *nextgreaters* that are computed during the construction of the timegraph and that indicate for each vertex v the nearest descendant v' of v on the same chain as v such that the graph entails $v < v'$. Each vertex (time point) in a timegraph has a *pseudotime* (or *rank*) associated with it, which is arbitrary except that it respects the ordering relationship between it and other vertices on the same chain. The main purpose of the pseudotimes is to allow the computation of the strongest relation entailed by the timegraph between two vertices on the same chain in constant time. In fact, given two vertices v_1 and v_2 on the same chain such that the pseudotime of v_2 is greater than the pseudotime of v_1, if the pseudotime of the nextgreater of v_1 is lower that the the pseudotime of v_2, then the timegraph entails $v_1 < v_2$, otherwise it entails $v_1 \leq v_2$.

Metric information such as absolute times (dates) can be represented by storing with each vertex two six-tuples of the form *(year month day hour minute second)*, the first of which corresponds to the earliest possible occurrence time and the second corresponds to the latest possible occurrence time. Each element of the tuple may be numeric or symbolic (e.g. *(1993 03 a 12 b c)* represents some time after 12 a.m. and before 1 p.m. of some day in March 1993).

Temporal distances (durations) between time points can be specified on each edge of the timegraph through a pair of numbers indicating the upper and lower bounds on the elapsed times (e.g., in seconds) between the time points represented by the vertices connected by the edge.

Task	Complexity
consistency	$O(e)$
ranks	$O(n+e)$
chains	$O(n+e+\hat{n}\cdot\hat{e})$
implicit $<$	$O(\hat{e}_{\neq}\cdot\hat{e})$

Table 1: Time complexity of the algorithms for the construction of a timegraph. e is the number of edges; $\hat{e}$ is the number of metaedges; n is the number of vertices; $\hat{n}$ is the number of metavertices; $\hat{e}_{\neq}$ is the number of cross-chain edges with label $\neq$

3 Building a timegraph

3.1 Qualitative information

Given a set S of relations in the Point Algebra we first build a TL-graph G whose vertices correspond to the variables (time points) in S, and whose edges correspond to the relations (note that "=" relations are translated by creating only one vertex for each set of explicitly equated variables, and labelling that vertex with that set of variables).

The construction of the timegraph from G consists of four main steps: consistency checking, ranking of the graph (assigning to each vertex a pseudotime), formation of the chains of the metagraph, and making explicit the implicit $<$ relations (see Figure 2). Table 1 reports the time complexity of the algorithms we have developed for achieving these tasks.[3]

Determining the consistency of G is accomplished by an $O(e)$ time algorithm based on two main steps: identifying all the strongly connected components (SCC) of the TL-graph derived from G ignoring the $\neq$ relations; checking if any of the SCCs contains a pair of vertices connected by an edge with label $<$, or $\neq$. When the graph is consistent, each SCC can be collapsed into any arbitrary vertex v within that component. All the cross-component edges entering or leaving the component are transferred to v. The edges within the component can be eliminated and a supplementary set of alternative names for v is generated.

The second step, ranking the graph, is accomplished by using a slight adaptation of the DAG-longest-paths algorithm [6] which takes $O(n+e)$ time. Originally we assigned pseudotimes independently on each chain. More recently, following [15] we have been using ranks as pseudotimes, ensuring that $rank(v) \leq rank(w)$ indicates the *absence* of paths from w to v even if v, w are on different chains.

The third step, the formation of the chains, consists of partitioning the TL-graph into a set of $\leq$-paths (time chains), deriving from this partition the metagraph and and doing a first-pass computation of the nextgreater links. The first two subtasks take time linear in $n+e$, while the last may require an $O(\hat{e})$ metagraph search for each of the $\hat{n}$ metavertices.

[3] A detailed description of the algorithms is available to the interested reader in [12].

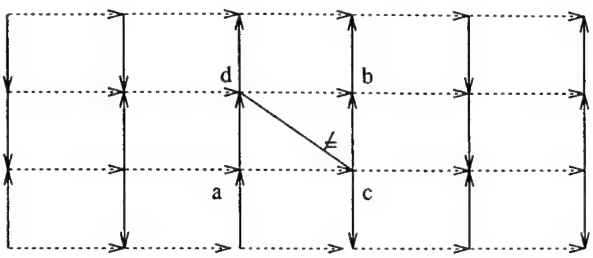


Figure 5: An example of timegraph with several $\neq$ diamonds of which only the one consisting of the vertices {a,b,c,d} is the smallest. A dotted arrow represents a $\leq$-path on a single chain

The fourth step, making explicit all the implicit $<$ relations, can be the most expensive task in the construction of a timegraph (see Table 1). This is the same as in van Beek's approach whose algorithm [31] takes $O(n^2 \cdot e_{\neq} + n^3)$ time, with n the number of time points, and $e_{\neq}$ the number of $\neq$ relations. Fortunately, the data structures provided by a timegraph allow this task to be accomplished more efficiently. A final operation is the revision of nextgreater links, to take account of any implicit $<$ relations made explicit in the fourth step.

Figure 2 shows the two cases of implicit $<$ relations. The time complexity of the algorithm we have developed for the first case is $O(\hat{e}_{\neq} \cdot (\hat{e} + \hat{n}))$.

In order to make implicit $<$ relations of the second kind (Figure 2.b) explicit, and removing redundant $\neq$-edges, a number of $\neq$ diamonds of the order of $e_{\neq} \cdot n^2$ may need to be identified in the worst case. However, for timegraphs only a subset of these needs to be considered. In fact it is possible to limit the search to the *smallest* $\neq$ diamonds, i.e., the set of diamonds obtained by considering for each edge $(v, \neq, w)$ only the *nearest common descendants* of v and w (NCD(v,w)) and their *nearest common ancestors* (NCA(v,w)). This is a consequence of the fact that, once we have inserted a $<$ edge from each vertex in NCA(v,w) to each vertex in NCD(v,w), we will have explicit $<$-paths for all pairs of "diamond-connected" vertices. From the previous observation we derive a criterion for pruning the search that in practice gives significant time savings.

Figure 5 shows an example of a timegraph with several $\neq$ diamonds of which only one is the smallest and needs to be considered.

3.2 Metric information

Globally optimal bounds on the absolute times of time points associated with vertices and locally optimal bounds on durations associated with edges are computed by constraint propagation without introduction of new edges. In essence, upper bounds are propagated backward and lower bounds forward. This ensures that each point has the best absolute time information possible.

The constraints used are inequalities relating the bounds associated with pairs of vertices connected by an edge. For example, if (l_1, u_1), (l_2, u_2) are the lower and upper bounds of the vertices 1 and 2 respectively, and (l, u) the lower and upper bounds on the duration of the edge from 1 to 2, then the inequality $l_2 - u \leq t_1 \leq u_2 - l$, where t_1 indicates the absolute time associated to the vertex 1, must be satisfied (among others) [30, 21].

4 Querying a timegraph

Given an explicit TL-graph we can derive the strongest relation between two time points that is entailed by the graph, by looking for all the paths connecting the two corresponding vertices. Furthermore, in timegraphs there are four special cases in which those paths can be obtained in constant time. Given two vertex labels p_1,p_2 the first case is the one where p_1 and p_2 are alternative names of the same point (see Section 2). The second case is the one where the vertices v_1 and v_2 corresponding to p_1, p_2 are on the same chain (see Section 2). The third case is the one where v_1 and v_2 are not on the same chain and have the same rank, and there is no $\neq$ edge between them (the entailed relation is $\{<, =, >\}$). The fourth case concerns $\neq$ relations for which the search is localized to the edges between the two input vertices. In fact, as a consequence of making implicit $<$ relations explicit, a $\neq$ relation can be the strongest relation entailed by the graph if and only if there exists a cross-edge with label $\neq$ connecting them.

In the remaining cases an explicit search of the graph needs to be performed. If there exists at least one $<$-path from v_1 to v_2, then the answer is $v_1 < v_2$. If there are only $\leq$-paths (but no $<$-paths) from v_1 to v_2, then the answer is $v_1 \leq v_2$. Analogously for paths from v_2 to v_1. An algorithm for accomplishing this task can be derived by an adaptation of the single-source-longest-paths algorithm reported in [6]. This algorithm has a time complexity of $O(h + \hat{e} + \hat{n})$, where h is the constant corresponding to the time required by the four special cases.

Querying metric information may require an $O(e)$ time search process. In fact, to determine the duration between any two points an exhaustive search must be done between these points to get the best duration bounds (the greatest minimum and the smallest maximum). Duration information on edges and implicit in absolute times is used. Details are given in [21].

5 Managing disjunctions

As briefly discussed in the introduction, the ability to manage disjunctions of qualitative temporal information is an important requirement in planning. For this reason we have extended our approach to include such disjunctions.

A *disjunctive timegraph* ($\mathcal{D}$-timegraph) is a pair $\langle T, D \rangle$ where T is a timegraph and D a set of PA-disjunctions involving only point-variables in T. A $\mathcal{D}$-timegraph $\langle T, D \rangle$ is *consistent*

if it is possible to select one of the disjuncts for each PA-disjunction in D in such a way that the resulting collection of selected PA-relations can be consistently added to T. We call this set of selected disjuncts an *instantiation* of D in T, and the task of finding such a set *deciding* D relative to T.

Once we have an instantiation of D, we can easily solve the problem of finding a consistent scenario by adding the instantiation to T and using a topological sort algorithm [32, 6]. Also the task of checking whether a relation R between two time points x and y is entailed by a $\mathcal{D}$-timegraph $\langle T, D\rangle$ can be reduced to the problem of finding an instantiation of D in an augmented version of T. We add the relation $x\overline{R}y$ to T (where $\overline{R}$ negation of R), obtaining a new timegraph T', and then check if (a) T' is consistent, and (b) D can be decided relative to T' (if not empty). The original $\mathcal{D}$-timegraph entails xRy just in case one of (a),(b) does not hold.

Deciding a set of binary disjunctions is an NP-complete problem [14]. However, recently we have developed a method for accomplishing this task which in practice performs very efficiently [11]. Given a disjunctive timegraph $\langle T, D\rangle$, the general method we have developed for deciding D relative to T consists of a two phases:

- a preprocessing phase which prunes the search space by reducing D to a subset D' of D and producing a timegraph T' such that D has an instantiation in T if and only if D' has an instantiation in T';
- a search phase which finds an instantiation of D' in T' (if it exists) by using backtracking.

Preprocessing uses a set of *pruning rules* exploiting the information provided by the timegraph to reduce the set of the disjunction[4]. This phase takes polynomial time, and in principle is strong enough to subsume consistency testing for Nebel and Bürckert's ORD-Horn subclass of interval relations [24].[5] The search phase is based on a form of *selective backtracking* (for more details see [12]), and uses a "forward propagation" technique to greatly enhance efficiency.

Using this method we have incorporated into TG-II a specialized algorithm for deciding a set of binary disjunction of inequalities (i.e., relations of the form $(pt1 < pt2) \vee (pt3 < pt4)$). Such disjunctions extend the set of interval relations which are translatable in terms of point relations to include, in particular, "before or after" (interval disjointness), and exclusion of a point p_1 form an interval with endpoints p_2 and p_3 (indicated below as $p_1 <\!\!\downarrow_{p_3}^{p_2}$), where $p_2 < p_3$. In fact, we have:[6]

$$I\,\{B\ A\}\ J \Leftrightarrow (I.end < J.start) \vee (J.end < I.start);$$

[4]For example, the "T-derivability" rule says that if the timegraph entails one of the disjuncts of a certain disjunction, then such a disjunction can be removed without loss of information.

[5]The Ord-Horn subclass of IA is a maximal tractable subalgebra of IA.

[6]Ordering relations between starting point and endpoint of intervals are omitted.

$$p_1 <\downarrow_{p_3}^{p_2} \Leftrightarrow ((p_1 < p_2) \vee (p_3 < p_1)) \wedge p_2 < p_3.$$

Moreover, some 3-interval and 4-interval relations not belonging to IA can be also stipulated, for example:

$$(I \{B\} J) \text{ or } (H \{B\} K) \Leftrightarrow (I.end < J.start) \vee (H.end < K.start).$$

Experimental results have shown that our algorithm is very efficient especially when the number of disjunctions is limited with respect to the number of the PA-relations in the timegraph, and that although consistency testing is NP-complete, in practice it tends to perform polynomially (see Section 7).

Future work concerns the extension of the algorithm in order include disjunctions of arbitrary PA-relations.

6 TimeGraph I and II

TimeGraph I (TG-I) and *TimeGraph II* (TG-II) are two temporal reasoning systems based on the representation and algorithms described in the previous Section. TG-I was originally developed in the context of natural language comprehension and has been integrated into a series of knowledge representation and reasoning systems, the two most recent being ECO-LOGIC [27] and EPILOG [26, 16]. TG-II is an extension of TG-I which is still in progress and which aims at generalizing the timegraph approach to a wider class of applications including planning and scheduling. Currently the most important extensions of TG-I which are incorporated into TG-II concern:

- the inclusion of algorithms for handling $\neq$ relations which are not handled by TG-I and which allow the representation of a larger class of interval relations;
- the inclusion of algorithms for deciding a set of of binary disjunctions of inequalities relative to the timegraph;
- an optimization of the chain partitioning of the TL-graph built from an initial set S of point relations. In principle, many logically equivalent timegraphs may be constructed from S since there can be many different ways to partition the TL-graph. TG-II attempts to minimize the number of chains and cross-edges;
- new efficient algorithms for computing the nextgreaters and for querying the timegraph, which use the rank to prune the search in the metagraph.

The most important features of TG-I which have not been integrated into TG-II yet are the management of metric information and the dynamic addition of new relations to the timegraph.

Both TG-I and TG-II are written in Common Lisp. TG-II is available to the interested reader via FTP from `cs.rochester.edu:/pub/knowledge-tools` files `tg-ii*`.[7]

6.1 The user interface of TG-I and TG-II

Both TG-I and TG-II have a user interface which allows the assertion of relations between time points as well as between time intervals called *episodes* in TG-I.[8]

The current TG-II's interface allows the assertion of all the relations in PA and of all the 188 interval relations of SIA [19, 33] (see Figure 6) extended with relations expressing point-interval exclusion and disjointness relations such as "before or after", which are representable through binary disjunctions of inequalities (see Section 5).

TG-I accepts assertions of relations in a subalgebra of PA called Convex Point Algebra (CPA) in [34] (i.e. the set of relations of PA with the exception of $\neq$), and of interval relations in a subclass of 83 relations of SIA (written as CSIA) which are listed in [33]. It also accepts assertions about absolute times and durations. On assertion, if an argument is an absolute time instead of a named time point or an interval, the appropriate absolute time of the other argument is updated. For example, *(I before (date '1993 '04 '01 '00 '00 '00))* would update the upper bound of I's absolute time (the maximum). For evaluation, the appropriate bound is compared against the absolute time, or two absolute times may be compared.

In addition, some predicates have been introduced to handle durations: *at-most-before,at-most-after,at-least-before,at-least-after, exactly-before,* and *exactly-after.*

These take three arguments, of which the first two may be intervals or time points (no absolute times) and the third denotes the duration between the two in seconds. *At-most-* implies maximum duration, *at-least-* implies minimum duration, and *exactly-* involves both (see Tables 2, 3 and 4).

Listed below are some of the main of user-interface functions[9] recognized by TG-I and TG-II (the strongest entailed relation between two points pt1 and pt2 (intervals I1 and I2) is indicated with SER(pt1,pt2) (SER(I1,I2)); optional arguments are enclosed by $\langle\ \rangle$.

- **(make-event** arg**)** [TG-I]
  This functions creates an interval named *arg* and asserts *arg.start* $<$ *arg.end*, where *arg.start* is the starting point of *arg* and *arg.end* is the endpoint. Make-event is defined only in TG-I, since in TG-II time points and intervals are automatically created when relations are asserted.

[7]The current version of TG-II available via FTP does not include the algorithms for managing disjunctions.

[8]An Interval is represented as a pair of strictly ordered points corresponding to the starting and end time of the interval.

[9]The actual implementation of most of them is in form of a Lisp "macro".

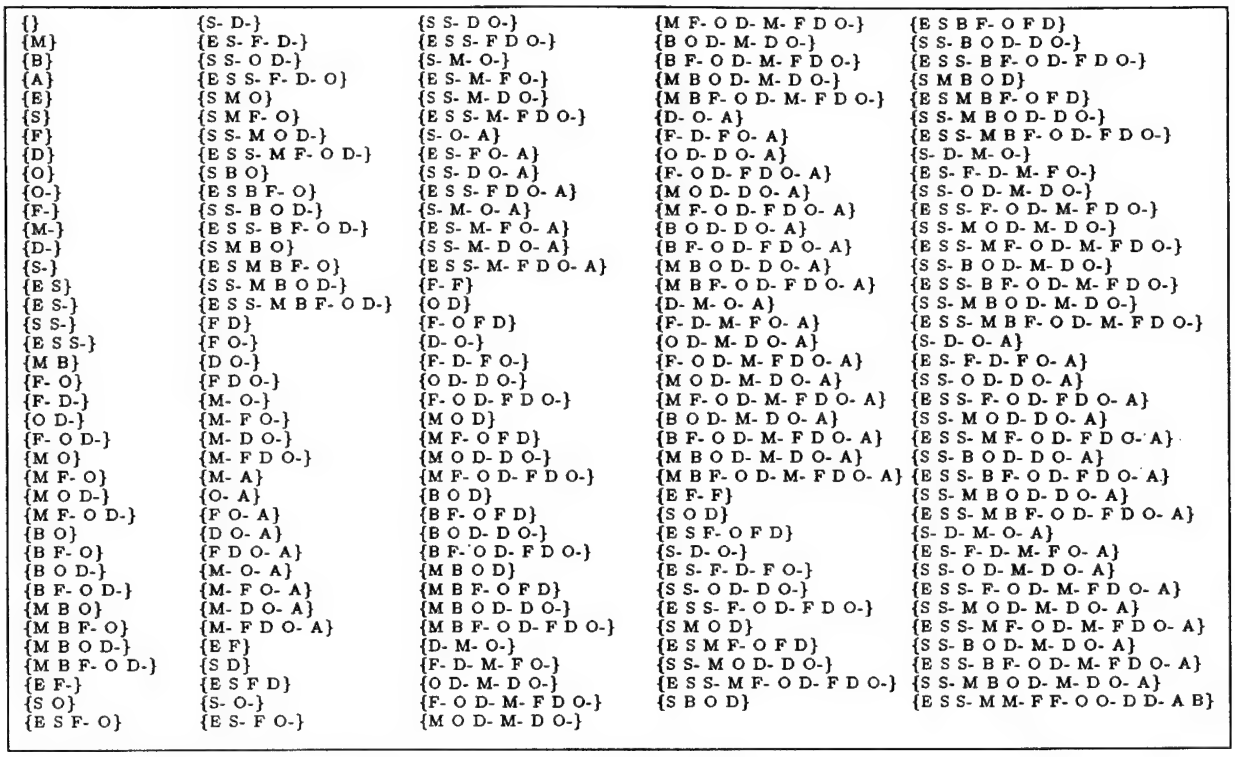


Figure 6: The interval relations of SIA

- (**asserta** arg1 reln arg2) [TG-I/II]
  This function is present both in TG-I and TG-II. It asserts that the relation between *arg1* and *arg2* is *reln*. For a description of the relations accepted by TG-I see Table 2. In TG-II *arg*1 and *arg*2 must be intervals,[10] and *reln* one of the interval relations of SIA (see Figure 6) extended by interval relations which can be expressed in terms of binary disjunctions of inequalities (see Section 5.)

- (**assertpr** pt1 reln pt2) [TG-II]
  This function asserts that the relation between *pt*1 and *pt*2 is *reln*, where *reln* is one of: $=, <, >, \leq, \geq, \neq$.

- (**make-tg**) [TG-II]
  This function builds the timegraph from the set of the stipulated relations. Note that in TG-I the timegraph is always built incrementally.[11]

- (**decide-disjunctions** *D*) [TG-II]
  This function decides a set *D* of disjunctions relative to the timegraph. In the current implementation *D* can be a set of binary disjunctions of inequalities only.

[10]Point relations can be expressed using the function "assertpr".

[11]In TG-I the timegraph is automatically updated after each assertion.

Argument1 Sort	Relation	Argument2 Sort	Argument3 Sort
episode	**equal**	*episode*	
point	**same-time**	*point*	
episode	**CSIA-relation**	*episode*	
point	**CPA-relation**	*point*	
episode	**between**	*episode*	*episode*
point		*point*	*point*
episode	**at-most-before**	*episode*	*number*
point	**at-least-before**	*point*	
	exactly-before		
	exactly-after		
	at-most-after		
	at-least-after		
episode	**has-duration**	*number*	

Table 2: The relations accepted by TG-I. In all patterns except the second last group (with at-most...), a *point* argument may be either a named time point or an absolute time specification. For the second last group, a point argument must be a named time point

- (**reln** arg1 arg2) [TG-I]
  This function returns SER(*arg*1,*arg*2), where *arg1* and *arg2* may be episodes, time points, or absolute times.

- (**elapsed** pt1 pt2) [TG-I]
  This function returns the best duration bounds between two time points (*pt1* and *pt2*) as a quoted expression of the form *(minimum maximum).*

- (**duration-of** e1) [TG-I]
  This function returns the duration bounds between the start of an episode and its end as a quoted expression of the form *(minimum maximum).*

- (**prel** ⟨pt1 ⟨pt2⟩⟩) [TG-II]
  When both the arguments are present this function returns the SER(pt1,pt2); when only pt1 is given it returns SER(pt1,x) for all the time points x of the timegraph; when none of the arguments is present it returns SER(x,y) for all the time points x and y of the timegraph. If the timegraph has not been built yet, the set of the stipulated relations is printed.

Relation	Meaning (e1 relation e2 e3)
between-0-0	$e1.end = e2.start$ and $e2.end = e3.start$
between-1-0	$e1.end < e2.start$ and $e2.end = e3.start$
between--0	$e1.end \leq e2.start$ and $e2.end = e3.start$
between-0-1	$e1.end = e2.start$ and $e2.end < e3.start$
between-0	$e1.end = e2.start$ and $e2.end \leq e3.start$
between-1-1	$e1.end < e2.start$ and $e2.end < e3.start$
between-1	$e1.end < e2.start$ and $e2.end \leq e3.start$
between--1	$e1.end \leq e2.start$ and $e2.end < e3.start$
between	$e1.end \leq e2.start$ and $e2.end \leq e3.start$

Table 3: The between-relations of TG-I. *e.start* and *e.end* indicate respectively the starting point and the endpoint of the interval (episode) *e*; "1" indicates strict (<); "0" indicates equal (meets), and nothing indicates non-strict ($\leq$)

Relation	Meaning (e1 relation {e2} d)
at-most-before	*e1* {*B M*} *e2* and maximum duration between them is **d**
at-least-before	*e1* {*B*} *e2* and minimum duration between them is **d**
exactly-before	*e1* {*B*} *e2* and both maximum **and** minimum durations are **d**
at-most-after	*e1* {*A M-*} *e2* and maximum duration between them is **d**
at-least-after	*e1* {*A*} *e2* and minimum duration between them is **d**
exactly-after	*e1* {*A*} *e2* and both maximum **and** minimum durations are **d**
has-duration	*e1.start before e1.end* and duration of *e*1 is **d**

Table 4: The metric relations of TG-I

- (**arel** ⟨I1 ⟨I2⟩⟩) [TG-II]
  When both the arguments are present this function returns the SER(I1,I2); when only I1 is given it returns SER(I1,x) for all the time intervals x in the timegraph; when none of the arguments is given it returns the SER(x,y) for all the time intervals x and y in the timegraph. If the timegraph has not been built yet, the set of the stipulated relations is printed.

- (**less-than** pt1 pt2) [TG-II]
  This function returns `T` when SER(pt1,pt2) is <; it returns `NIL` when SER(pt1,pt2) is $\geq$; it returns `unknown` in the other cases.

- (**less-or-equal** pt1 pt2) [TG-II]
  This function returns `T` when SER(pt1,pt2) is $\leq$; it return `NIL` when SER(pt1,pt2) is >; it returns `unknown` in the other cases.

- (**greater-than** pt1 pt2) [TG-II]
  This function returns T when SER(pt1,pt2) is $>$; it return NIL when SER(pt1,pt2) is $\leq$; it returns unknown in the other cases.

- (**greater-or-equal** pt1 pt2) [TG-II]
  This function returns T when SER(pt1,pt2) is $\geq$; it returns NIL when SER(pt1,pt2) is $<$; it returns unknown in the other cases.

- (**equal-to** pt1 pt2) [TG-II]
  This function returns T when SER(pt1,pt2) is $=$; it returns NIL when SER(pt1,pt2) is $<,>$ or $\neq$; it returns unknown in the other cases.

- (**not-equal-to** pt1 pt2) [TG-II]
  This function returns T when SER(pt1,pt2) is $<,>$ or $\neq$; it returns NIL when SER(pt1,pt2) is $=$; it returns unknown in the other cases.

- (**pointizable** reln)
  This functions returns T if *reln* is a pointizable interval relation, i.e. it is an interval relations of SIA.

- (**print-tg**) [TG-II]
  This function prints the following information about the timegraph and the metagraph: the length, the starting point and the endpoint of each chain; the nextgreater and the prevless links of each vertex; all the metavertices (cross-connected vertices) with the corresponding metaedges (cross-edges).

7 TG-II's performance

In this Section we report some results from large scale tests we have conducted with TG-II on a SUN SPARCstation 10. Table 5 provides statistics about building timegraphs for randomly generated consistent sets of relations. For each number n of time points considered, the test procedure generates n sets of relations, each containing a number of relations equal to $n \cdot INT(log_2 n)$. For example, with $n = 500$, the procedure generates 500 timegraphs from 500 data sets each of which contains 4000 relations. The table provides the average CPU-time (milliseconds) for building the timegraph, the average number of chains, the average length of the chains and the average maximum length of the chains.

Given a set S of n time points with indices $1, ..., n$, the first m points (S') – with m a random number between 1 and n – are chosen to represent distinct elements whose time order is the same as the order of their indices; the remaining $n-m$ points (S'') are randomly assigned to coincide with points in S'. Relations are generated randomly by choosing a pair i, j ($i < j$) of indices. Specifically, if both i and j are among the m distinct points, then the relation iRj is generated, where R is randomly taken from $\{<, \leq, \neq\}$, with the percentage

points	CPU-time	chains	length-chain	max-chain
100	74	20.3	6.10	48.9
200	191	40.5	8.18	97.5
300	308	54.6	12.21	154.1
400	470	73.3	12.28	206.2
500	592	96.5	11.47	244.7
1000	1711	182.8	14.36	508.8

Table 5: Statistics about the construction of a timegraph

of $\neq$ relations kept low (2%). If i or j (or both) is one of the $n - m$ points in S'', the corresponding point in S' is considered. For example, if $n = 100$, $m = 70$, $i = 50$, $j = 80$ and j was assigned to 50, then a relation between i and j is randomly taken from $\{\leq, \geq, =\}$. Since we were mostly interested in measuring TG-II's performance with data sets likely to allow chain formation, the pairs of points were generated using a geometric distribution with expected value 3.

Other experiments show that querying a timegraph is on average a fast operation. For example, the average CPU-time over 100,000 queries on 100 randomly generated timegraphs with 500 points was 2.8 milliseconds.

The curves in Figure 7 report the results of an experiment aimed at testing the scalability of TG-II for the task of deciding randomly generated consistent set of disjunctions (see [12] for more details). The two curves of the first graph show the average CPU-time (seconds) required for deciding sets of disjunctions of size n (for n the number of time points) with $8n$ and $2nlogn$ simple PA-relations (with the logarithm truncated to its integer part). The numbers attached to the points on the curve indicate the percentage of the disjunctions eliminated by preprocessing. While in the case of $8n$ relations this percentage decreases when n increases, for $2nlogn$ relations it tends to be constant (wavering between 88.8 and 91.3).[12]

The curve in the second graph shows the average CPU-time required by more sparse graphs ($4n$ PA-relations) for which forward propagation has been used during the search.[13]

The third graph shows that for the 16,000 problems considered our algorithms tend to perform polynomially (quadratic time for the curves of the first graph and cubic time for the curve of the second) since the previous curves approximate straight lines on a log–log scale.

Finally, we should mention some experiments conducted by Yampratoom and Allen [37] comparing the performance of TimeGraph I and II with several other temporal reasoning systems – TimeLogic [18], MATS [17], Tachyon [5] and TMM [7] – in which the timegraph

[12]In order to simplify the figure these numbers are not shown.

[13]These experiments were conducted on a SUN SPARCstation 2 and hence a comparison with the results in the previous graph is inappropriate.

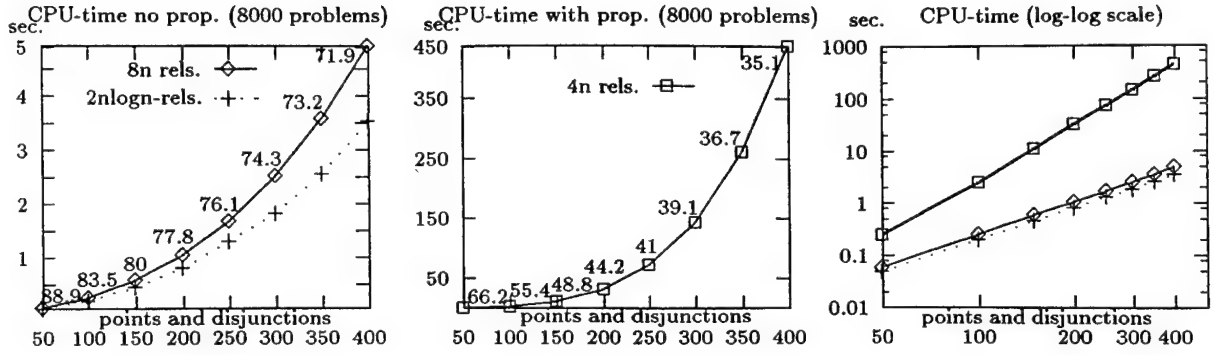


Figure 7: Scalability of the algorithms for deciding a set of disjunctions

approach proved by far the most efficient for large data sets generated for the TRAINS world [1].

8 Conclusions and future work

We have described two efficient domain-independent temporal reasoning systems. Our approach is related to others recently proposed [9, 15] and whose main characteristic is that of providing better performance in practice compared to the more traditional constraint-based approaches.

In particular, in our systems both space and time complexity are reduced significantly for many practical applications. In fact, the space complexity depends on the size of the set of the stipulated relations and on a limited number of implicit $<$-relations (those araising from $\neq$ diamonds), and not on the number of time points (as in the constraint propagation approach). Instead of computing the closure of the set of relations in PA or SIA, we build timegraph providing a collection of data structures that allow efficient deduction of relations at query time.

TG-I can handle the qualitative relations of CPA and CSIA, and metric information such as durations and absolute times. TG-II considerably extends the expressiveness of TG-I at the qualitative level by including efficient algorithms for managing $\neq$-relations and binary disjunctions of inequalities, allowing the representation of all the interval relations of SIA, as well as point-interval exclusion and interval disjointness. Experimental results have shown that TG-I and TG-II are much faster than other systems based on constraint propagation.

Current and future work on TG-II concerns the design of efficient algorithms for adding new relations to the timegraph dynamically, the integration of metric relations,[14] and the

[14]These were handled in TG-I, but as noted $\neq$ was not handled and also $<$ and $\leq$ relations entailed via metric relations were not extracted in a deductively complete way.

design and implementation of an efficient algorithm for handling disjunctions of arbitrary PA-relations.

Acknowledgements

The work of the first author was carried out in part during a visit at the Computer Science Department of the University of Rochester supported by the Italian National Research Council (CNR). The second author was supported by Rome Lab Contract F30602-91-C-0010, and the third by Boeing Contract W 297884.

9 References

[1] J. Allen and L. Schubert. The trains project. Technical Report 91-1, Department of Computer Science, University of Rochester, Rochester, NY, 1991.

[2] J. F. Allen. Maintaining knowledge about temporal intervals. *Communication of the ACM*, 26(1):832–843, 1983.

[3] J. F. Allen, H. Kautz, R. Pelavin, and J. Tenenberg. *Reasoning about Plans.* Morgan-Kaufmann, San Mateo, CA, 1991.

[4] J. F. Allen and J. A. Koomen. Planning using a temporal world model. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 741–747, Karlsruhe, Germany, 1983.

[5] R. Arthur and J. Stillman. Temporal reasoning for planning and scheduling. user's guide: Preliminary realise. Technical report, Artificial Intelligence Laboratory, General Electric Research and Development Center, 1992.

[6] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[7] T. Dean and D. V. McDermott. Temporal data base management. *Artificial Intelligence*, 32:1–55, 1987.

[8] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[9] J. Dorn. Temporal reasoning in sequence graphs. In *Proceedings of the Tenth National Conference of the American Association for Artificial Intelligence (AAAI-92)*, pages 735,740, 1992.

[10] A. Gerevini and L. Schubert. Efficient temporal reasoning through timegraphs. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 648–654, Chambéry, Savoie, France. 1993. Morgan-Kaufmann.

[11] A. Gerevini and L. Schubert. An efficient method for managing disjunctions in qualitative temporal reasoning. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR94)*, pages 215–225, San Francisco, CA, 1994. Morgan-Kaufmann.

[12] A. Gerevini and L. Schubert. Efficient algorithms for qualitative reasoning about time. *Artificial Intelligence*, 1995. To appear. Also available as: IRST Technical Report 9307-44, Istituto per la Ricerca Scientifica e Tecnologica, 38050 Povo, Trento Italy; Technical Report 496, Computer Science Dept., University of Rochester, Rochester, NY 14627. USA.

[13] A. Gerevini and L. Schubert. On computing the minimal labels in time point algebra networks. *Computational Intelligence*, 1995. To appear. Also available as: IRST Technical Report 9408-10, Istituto per la Ricerca Scientifica e Tecnologica, 38050, Povo Trento, Italy; Technical Report 495, Computer Science Dept., University of Rochester. Rochester, NY 14627, USA.

[14] A. Gerevini and L. Schubert. On point-based temporal disjointness. *Artificial Intelligence.* 70:347–361. 1994.

[15] M. Ghallab and A. Mounir Alaoui. Managing efficiently temporal relations through indexed spanning trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 1297–1303, 1989.

[16] C. H. Hwang and L. H. Schubert. Meeting the interlocking needs of lf-computation, deindexing, and inference: An organic approach to general nlu. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, Chambéry, Savoie, France, 1993.

[17] H. Kautz and P. Ladkin. Integrating metric and qualitative temporal reasoning. In *Proceedings of the Nineth National Conference of the American Association for Artificial Intelligence*, Anheim, Los Angeles, CA, 1991.

[18] Johannes A. G. M. Koomen. The timelogic temporal reasoning system. Technical Report 231, University of Rochester, Computer Science Dept. Rochester New York 14627, 1988.

[19] P. Ladkin and R. Maddux. On binary constraint networks. Technical Report KES.U.88.8, Kestrel Institute, Palo Alto, CA, 1988.

[20] P. Ladkin and R. Maddux. On binary constraint problems. *Journal of the Association for Computing Machinery (ACM)*, 41(3), 1994.

[21] S. A. Miller. Time revised. Technical Report M.Sc. thesis, Dept. of Computer Science, The University of Alberta, Edmonton, Alta, 1988.

[22] S. A. Miller and L. K. Schubert. Time revisited. *Computational Intelligence*, 6:108–118, 1990.

[23] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7(3):95–132, 1974.

[24] B. Nebel and H. J. Bürckert. Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra. In *Proceedings of the Eleventh National Conference of the American Association for Artificial Intelligence (AAAI-94)*, Seattle, WA, 1994. Morgan-Kaufmann.

[25] J.S. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for adl. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–114, 1992.

[26] S. A. Schaeffer, C. H. Hwang, J. de Haan, and L. K. Schubert. The user's guide to EPILOG. Technical report, Dept. of Computing Science, Univ. of Alberta, Edmonton, Alberta, Canada, 1991.

[27] L. K. Schubert and C. H. Hwang. An episodic knowledge representation for narrative texts. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 444–458, Toronto, Ont., 1989.

[28] L. K. Schubert, M. A. Papalaskaris, and J. Taugher. Determining type, part, colour, and time relationships. *Computer* (special issue on Knowledge Representation), 16:53–60, 1983.

[29] L. K. Schubert, M. A. Papalaskaris, and J. Taugher. Accelerating deductive inference: Special methods for taxonomies, colours, and times. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 187–220. Springer-Verlag, Berlin, Heidelberg, New York, 1987.

[30] J. Taugher. An efficient representation for time information. In *M.Sc. thesis*, Department of Computer Science, University of Alberta, Edmont, Alta, 1983.

[31] P. van Beek. Reasoning about qualitative temporal information. In *Proceedings of the Eighth National Conference of the American Association for Artificial Intelligence (AAAI-90)*, pages 728–734, Boston, MA, 1990.

[32] P. van Beek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58(1-3):297–321, 1992.

[33] P. van Beek and R. Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–144, 1990.

[34] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the Fifth National Conference of the American Association for Artificial Intelligence (AAAI-86)*, pages 377–382, Philadelphia, PA, 1986.

[35] M. Vilain, H. Kautz, and P. van Beek. Constraint propagation algorithms for temporal reasoning: a revised report. In *Readings in Qualitative Reasoning about Physical Systems*, pages 373–381. Morgan Kaufmann, San Mateo, CA, 1990.

[36] D. S. Weld. An introduction to least commitment planning. *AI Magazine*, Fall, 1994.

[37] E. Yampratoom and J. Allen. Performance of temporal reasoning systems. *SIGART Bulletin*, 4(3), 1993.

Appendix H

A Decision Theoretic Planning Assistant*

Abstract

We discuss three difficulties in applying decision theory directly to planning: the computational complexity of reasoning about probability, the difficulty of gathering knowledge of probability and the difficulty of specifying behavior using utilities. We then describe a decision theoretic planning assistant that avoids some of these problems by assisting a traditional planner instead of generating plans. It reasons about probability based on its experience, so does not require precise probabilities supplied in advance and its behavior is specified through goals.

1 Introduction

Traditional planners operate in restricted domains in which knowledge is certain and a single goal is specified. In realistic situations, planners must be able to balance achieving several goals of lower priority against achieving a single high priority goal. Without multiple prioritized goals, a planner will produce the best plan even if that plan is terrible. For example, if a planner knows only about trains, it will build a plan to ship oranges to Hawaii by rail. Also, unless it attends to circumstances unrelated to their goals, it may achieve its goals at an unacceptable cost. For example, a robot railroad engineer will speed unless it knows that losing an engine is worse than missing a schedule. Because traditional planners are driven only by their goals, they cannot decide to forego a goal that can be achieve only at unacceptable cost.

*This material is based on: Nat Martin and James Allen, "A Decision Theoretic Planning Assistant," *Proceedings of the International Conference on Statistics and Artificial Intelligence*, Fort Lauderdale, FL, January, 1993.

Decision theory computes a preference order over competing goals of differing priorities and probabilities promising a significant increase in the expressiveness of planning languages. It allows for a unified treatment of multiple prioritized goals and uncertainty. Unfortunately, this increased expressiveness is costly.

1.1 Constraints on Decision Theory in Planning

Though decision theory allows one to encode information about uncertainty using probability and to encode information about multiple goals using utilities, its use in planning systems is complicated by three factors:

1. reasoning about probability is computationally complex;
2. necessary probabilities are often unknown; and
3. the relationship between utility and behavior is not well understood.

The computational complexity of reasoning about probability is high. Though Bayes nets have limited expressiveness (they can only encode propositional knowledge), the computational complexity of inferring probabilities using them is in NP [Cooper, 1987]. Because first-order logics of probability are more expressive, the complexity of inferring probabilities using them is worse than undecidable [Abadi and Halpern, 1989].

Decision theoretic planners can reduce the complexity of their task by encoding the STRIPS assumption in independence assumptions [Wellman, 1990]. Such planners can use Bayes nets [Pearl, 1988] to provide a graphical method of representing these assumptions, allowing one to encode reasonable independence assumptions for probabilistic systems. Unfortunately, these techniques only minimize the increased cost of decision theory; they do not eliminate it.

The precise knowledge of probability required for decision theory also constrains its use. Probability can be understood as a representation of human uncertainty. In areas like medicine where articulate experts have access to statistical data, it can represent these experts' uncertainty. Unfortunately, such subjective probabilities are impractical in domains lacking articulate experts. As an alternative to asking experts, probabilities can be estimated from observations. For example, no one knows the probability that customers leave things under tables in fast food restaurants, though robots designed to clean such restaurants might benefit from this information. The robot cleaning the floor does have the opportunity to gather information about the likelihood of things being left on the floor. From these observations, it could estimate the desired probability. It can also use information about the precision of the estimates it has computed.

Though recent work [Haddawy and Hanks, 1990; Wellman and Doyle, 1992] explores the relationship between goals and utilities, specifying a utility function that will cause a planner to behave properly is still poorly understood. When we specify a goal to a traditional

planner, we know something about the resulting state of the world in which we execute a successful plan, namely that the goal will hold. We know nothing about the state of the world after the execution of a decision theoretic planner's plan. We may want our planner to achieve many things, but a successful decision theoretic plan may do nothing because the risk action is too great. This uncertainty about the state of the world after a decision theoretic planner executes its plan makes it hard to design the utilities so a planner will achieve something concrete. It also makes it hard to evaluate the plan produced. Indeed, the only reasonable evaluation is performing the same analysis the planner did.

We have built a decision theoretic planning assistant that collects statistics and suggests the actions that are most likely to achieve effects given assumptions about the state of the world. In doing so, it avoids the computational complexity of generating and evaluating plans and it requires no prior knowledge of probability. It also treats each event mentioned in the plan it executes as single goals allowing a simple utility function.

2 A Decision Theoretic Planning Assistant

The Decision Theoretic Planning Assistant (DTPA) was constructed using an extension of Rhet [Martin, 1993]. This planning assistant provides three services to a traditional planner:

1. it advises the planner about the likelihood of events;
2. it executes plans by choosing the actions most likely to accomplish the planner's goals; and
3. it monitors the actions to insure that they have the effects the traditional planning module hypothesized and notifies the planner when a low probability event causes the plan to go awry.

The planning assistant's knowledge of probability is based on the number of instances of event types it has seen. The event types are arranged in a hierarchy such that all event instances of children are event instances of the parent. Therefore, the planning assistant has more accurate knowledge of the probability of event types higher in the hierarchy than about event types lower in the hierarchy.

2.1 The Language

Rhet [Allen and Miller, 1991] is a knowledge representation that provides, among other things, inference like Prolog's, a type system, and an equality reasoning system that allows function terms. To build the DTPA, Rhet was extended to represent probability, utility and the number of occurrences of events efficiently. The probability function is the confidence interval for the number of occurrences of a reference event type that are also occurrences of a

target event type. For example, if an engineer has been asked to load a car one hundred times, but has taken the appropriate action only fifty times, the probability at the .05 confidence level would be [.4 .6], (*i.e.*, the probability is somewhere between .4 and .6.)

Utilities are numbers and expected utilities are calculated by multiplying the lower bound of the confidence interval and the upper bound of the confidence interval by the utility. Therefore, like probabilities, expected utilities are intervals.

The language provides a function `best-event-type` that takes a confidence level, a set of possible actions event types and a goal event type as parameters. It returns the set of the possible action event types among which it cannot choose.

It also provides a function `execute` that takes a plan (*i.e.*, a set of event types and constraints on these event types) as an argument. This function sends requests to the agents who are most likely to be able to cause the events specified in the plan.

The DTPA represents causation by the predicate `might-cause`. This predicate focuses the DTPA attention on only those actions for which the programmer has explicitly stated might cause the goal. Because the goal is possible after any action, the DTPA considers too many actions if it relies only on probability.

2.2 Example

As a running example, consider a computerized planner trying to get some oranges from Avon to Bath. There is a rail line between Avon and Bath. In Avon, there is a warehouse with oranges, an engine, E1, and a boxcar, B1. The warehouse is managed by manager, W1. One plan that achieves this goal is: ask W1 to load the oranges into B1 then ask E1 to couple to B1, drive to Bath and unload the car. We will concentrate of the first step of the plan, loading B1 with oranges.

This example comes from the TRAINS domain [Allen and Schubert, 1991], a transportation domain containing engineers, managers of factories, warehouses and production centers. The DTPA was tested in this domain and this example was derived from one of those tests.

In the TRAINS domain, the DTPA does not itself perform physical actions. Instead, all of its actions are requests to appropriate agents to perform actions. For example, when the planning assistant chooses `W1-Load` to try to get oranges loaded into a railroad car, it will actually send a radio message (*i.e.*, a Lisp expression through a TCP/IP connection) to the agent requesting that the agent load the car. Agents send reports when they receive a request, when they initiate a requested action and when they complete the requested action. Actions are programs the agents execute; they may or may not have their intended effects. Therefore, any one of these reports may fail to reach the DTPA. If the planning assistant does not receive a report either the action could have failed or the report could have been lost.

2.3 Advising a Planner

All planners must make assumptions as they plan. Some early planning systems build their assumptions into their knowledge representation language. For example, STRIPS [Fikes and Nilsson, 1971] cannot represent events it does not cause. First-order languages can represent more complex interactions between actions but this increased expressiveness is bought at the cost of making inference difficult. STRIPS can generate plans because it assumes that its knowledge is complete and accurate. Planners built with more expressive languages cannot make that assumption so it is difficult to prove that the result of the plan will be the goal. For example, suppose the planner asks W1 to load oranges into B1, then asks E1 to take B1 to Avon. The oranges will get to Avon only if the oranges are in the car when E1 leaves and remain in the car until E1 arrives at Bath. Two possible problems arise. E1 might not wait until the oranges are loaded and E1 might not wait until it gets to Bath to unload the oranges. In the first case, the oranges remain in Avon; in the second, they end up on the tracks between Avon and Bath.[1]

Allen *et. al.* [1991] describe a planner that generates plans by proving that the planner's goals will hold if it executes its plan. The planner generates plans by searching through the space of plans, which is a space of assumptions about the future. The planner makes two types of assumptions about the future:

1. that it will be able to execute any action at any time and
2. that the effects of the actions it executes persist until they are needed.

If the planner can prove a contradiction using an assumption, it abandons that assumption and tries another. When it finds a set of assumptions that is consistent with the current state of the world and that allows it to prove its goal will hold, it uses the assumed actions under the assumed constraints as its plan.

The DTPA helps the planner choose appropriate assumptions. To do so, the planner supplies the planning assistant with a confidence level, a set of candidate event types, an event type that it wants to cause and a temporal interval. The planning assistant returns the set of event types most likely to cause the desired event type. The set of event types is those among which the planning assistant cannot choose. This indicates that any of these assumptions are equally valid as far as the planning assistant knows.

Example:

> In constructing its simple plan, the planner can choose W1 and E1 to load the oranges into B1. Unless the planner has explicit knowledge that one agent is preferable to the other, it cannot choose. The planning assistant can use it knowledge of previous occurrences of requests to couple to make the choice.
>
> Suppose we have three event types named:

[1] Robots are stupid. We forgot to tell one of our simulated engineers to wait until it got to the station to unload its oranges and the oranges ended up on the track. It took quite awhile to find them too.

1. E1-Load
2. W1-Load
3. Loaded

The first is the set of event instances that include a request to E1 to load a car, the second the set of event instances that include a request to W1 to load a car and the third the set of event instances in which a car is loaded. The planner and the planning assistant know that either request might result in the cars being coupled.

```
[might-cause E1-Load Loaded]
[might-cause W1-Load Loaded]
```

Which engine should the planner try to couple to the car?

The planner asks the DTPA for its suggestion using `best-event-type` as follows

```
(best-event-type .95 Loaded
           E1-Load W1-Load)
```

Depending on the statistics and asserted probabilities, this function call will return either `E1-Load`, `W1-Load`, or both.

Given the statistics below, `best-event-type` returns `W1-Load`.

```
[occ so-far 100 E1-Load]
[occ so-far 35 E1-Load Loaded]
[occ so-far 100 W1-Load]
[occ so-far 65 W1-Load Loaded]
```

On the other hand, if the number of occurrences are different, calling `best-event-type` returns `(W1-Load E1-Load)`.

```
[occ so-far 100 E1-Load]
[occ so-far 50 E1-Load Load]
[occ so-far 100 W1-Load]
[occ so-far 60 W1-Load Load]
```

In the second case the DTPA has no useful information.

The DTPA can answer questions about probability quickly because the planner constrains the question. It gives the DTPA the target event type and a set of reference event-types. It is easy for the DTPA to make decision under such constrained circumstances.

2.3.1 Persistence assumptions

Persistence assumptions are assumptions that properties the planner knows to be true now will continue to be true. They are common in planning because, even if the planner knows it caused an event that made a property true, it does not know that the property remains true until that property is needed. For example, assuming that oranges remain in the railroad car until they get to their destination is a persistence assumption.

One can describe event types in which the event instances of the event types are separated by another event instance. In the previous example, we describe an event type that consisted of event instances that were constructed from one event type following another, we could describe such an event type consisting of three event instances: the two we are interested in and a third that separates the other two.

Example:

Suppose the planning assistant's knowledge base is the same as in the previous example. In addition, the planner has two additional event types called `Load-Couple` in which loading the oranges is followed by coupling the loaded car to the train and `Load-Event-Couple` in which some unspecified event occurs between the loading and the coupling

The planner describes this event using the following Rhet axioms, which are Horn clauses where the head is separated from the rest of the clause by the symbol "<". The first states that an instance of the `Load-Couple` action event type occurs when an instance of the `Load` action event type occurs before an instance of a `Couple` event type. The second states that an instance of the `Load-Event-Couple` type occurs when an instance of the `Load` event type occurs, then any event instance occurs and finally an instance of the `Couple` event type occurs.

```
[[element-of ?ei_3 Load-Couple] <
 [element-of ?ei_1 Load]
 [element-of ?ei_2 Couple]
 [Time-Before [time ?ei_1] [time ?ei_2]]]

[[element-of ?ei4 Load-Event-Couple] <
 [element-of ?ei1 Load]
 [element-of ?ei2 ?et]
 [element-of ?ei3 Couple]
 [Time-Before [time ?ei1] [time ?ei2]]
 [Time-Before [time ?ei2] [time ?ei3]]]
```

The planner also knows that either of these event types might move the oranges.

```
[might-cause Load-Couple Move]
[might-cause Load-Event-Couple Move]
```

The plan would then call `best-event-type` as follows.

```
(best-event-type .95 Move
          Load-Couple Load-Event-Couple)
```

Depending on the statistics, this function call will return either

`Load-Couple`, `Load-Event-Couple`, or both. Here, if the confidence intervals overlap, the planner knows that the planning assistant had no information that contradicts the persistence assumptions necessary. Because the confidence intervals are incomparable, the planning assistant has no information indication that an event occurring between the loading and the coupling changes the probability of getting the commodity moved. Therefore, as far as the planning assistant knows, the properties necessary to move the oranges persist through the occurrence of any event.

Suppose the number of occurrences is as below.

```
[occ so-far 100 Load-Event-Couple]
[occ so-far 35 Load-Event-Couple Move]
[occ so-far 100 Load-Couple]
[occ so-far 65 Load-Couple Move]
```

Here, calling `best-event-type` returns the event type `Load-Couple`. The planner would know that it is important whether an event instance occurs between the load of the oranges and the coupling of the engine.

Suppose, on the other hand, that the number of occurrences is different.

```
[occ so-far 100 Load-Event-Couple]
[occ so-far 50 Load-Event-Couple Move]
[occ so-far 100 Load-Couple]
[occ so-far 60 Load-Couple Move]
```

Calling `best-event-type`, as above, returns the set (`Load-Couple`

`Load-Event-Couple`). The planner can reason that it has insufficient information at the .95 confidence level to assume that an event instance separating the loading of the oranges and the coupling of the engine make the action fail to achieve its goal. The planner may choose to assume that the necessary properties for the event to hold continue to hold through any event instance it can describe.

2.4 Executing a Plan

The DTPA executes a plan by assuming that there are no interactions between the event types specified in the plan. It can make this assumption because the planner and the planning assistant work in the same domain of discourse. The planner will have considered all known interactions between the events; if there are other interactions, the DTPA will also be unaware of them. The assumption allows the planning assistant to deal with each event in the plan as a separate goal.

The DTPA treats the execution of a plan as a sequence of forced choices. It first tries to make the choice by comparing confidence intervals. It this fails, it abstracts the choice by removing constraints on the actions. If it cannot make a choice at an acceptable level of abstraction, it tries to make a choice using the maximum likelihood estimate. If this fails, it chooses at random.

2.4.1 Abstraction Hierarchy

The abstraction hierarchy has the set of all event instances at its root. Below the root are more specific sets of events. For example, all actions are event types that have an agent associated with them.

The DTPA chooses events to execute from the set of actions it can cause. This is a specialization of action events. The actions the DTPA can cause are further specialized by adding constraints onto them. For example, loading a car in Avon is a specialization of loading a car anywhere.

Defining event types is one of the most difficult tasks in writing programs for the DTPA. The task is difficult because describing even simple causal models requires many event types. The task is more difficult when one tries to arrange event types into a useful hierarchy.

The task of generating event hierarchies is difficult because the programmer controls the DTPA's learning process by defining event types. The DTPA can only learn probabilities about the event types, so if its programmer has not added a particular event type, the DTPA will not waste time considering it. This attribute is vitally important because the effectiveness of the DTPA's choices is constrained by the number of event types it must consider. If it must consider many event types, it will need a great deal of experience before it can say with certainty that one is better than all the others. On the other hand, if it chooses from only a few choices, a few examples may allow it to make a clear choice.

The DTPA uses the event type hierarchy by looking for the most specific event type that might cause the event type requested by the planner. If the plan specifies an event type in the planning assistant's event hierarchy, it searches for the action event type most likely to cause event type directly.

The situation is more complicated when the planning assistant's event hierarchy does not match the planner's. Here, the planning assistant chooses a random element of the planner's event type and searches for the most specific event type in its hierarchy that contains the random element. In doing this, the planning assistant assumes that the plan the planner has specified depends on any instance of the event type specified. The random element of the event type chosen will be such an instance, and any element of an event type containing the random element will also be such an instance. The properties that characterize the event type specified will also characterize the event type that the planner eventually chooses.

Once the planner has found appropriate event types, it assigns a constant utility to each of the event types so selected. It then uses the DTPA's search capabilities to look for the event type that it can cause that is most likely to cause that event type.

Example:

The following shows a part of the abstraction hierarchy for loading cars. All actions are events and all loadings are actions. At the next level the hierarchy splits between a manager loading and an engineer loading. The event instances in which a manager loading are further split into loading by the different managers.

```
Event
  Action
    Load
      Manager-Load
        W1-Load
            ...
      Engineer-Load
        E1-Load
            ...
```

The abstraction hierarchy is further elaborated by selectively removing preconditions to execution of the program associated with the action. The hierarchy below shows that W1 can load a car when the car and stuff to load are available, or it can ignore one or both of the preconditions. Every execution in which the precondition holds is also an instance of an execution in which the precondition may or may not hold.

```
W1-Load
  W1-Load(Here(Car))
```

```
  W1-Load(Here(Stuff), Here(Car))
W1-Load(Here(Stuff))
  W1-Load(Here(Stuff), Here(Car))
```

This type of abstraction is problematical. Because event types may appear more than once in the hierarchy (*e.g.*, `W1-Load(Here(Stuff), Here(Car))`), there may be more event types at the more abstract level than at the more concrete levels. If this occurs, abstracting make choices harder rather than easier.

2.4.2 Choosing an Event

The planning assistant causes each event specified in the plan by searching for the action event most likely to cause the specified event and executing the action associated with that event. It first selects the set the event types that `might-cause` the event specified in the plan. This set can be further restricted by the programmer. From the restricted set, it selects those events that it can cause (*i.e.*, requests for action), then chooses among this set based on the conditional probability of the specified event given those event types by choosing the event type with the highest such conditional probability.

The DTPA first chooses at the .95 α level, and if a choice can be made at that level, it is returned. If the choice function returns a set of candidates among which it cannot choose, the planning assistant abstracts the choice by removing some of the preconditions of the choice. This tack might be successful because the DTPA's set of observations may be spread among fewer event types making the confidence intervals narrower.

If abstraction does not help it choose based on confidence intervals, it chooses from the most specific set using the maximum likelihood estimate[2]. If the maximum likelihood estimates for the necessary conditional probabilities are equal, choice is impossible, so the planning assistant chooses randomly[3].

Example:

If the DTPA has the following statistics, it cannot decide at the most concrete level of abstraction. The probability intervals it generates are [0.37 0.63] for the concrete action involving E1 and [0.19 0.44] for the action involving W1.

[2]The planning assistant computes a maximum likelihood estimate using the formula $\frac{y+1}{n+2}$ where y is the number of times the event type under consideration has caused the desired event type and n is the number of times the event type under consideration has been observed. It uses this formula because it gives the lowest score to the event that has failed most often even if none has succeeded.

[3]It would be interesting to try choosing using the maximum likelihood estimate at different levels of abstraction before choosing randomly. At present it does not.

```
[occ so-far 50
  E1-Load(Here(Stuff), Here(Car))]
[occ so-far 15
  E1-Load(Here(Stuff), Here(Car)) Loaded]
[occ so-far 50
  W1-Load(Here(Stuff), Here(Car))]
[occ so-far 25
  W1-Load(Here(Stuff), Here(Car)) Loaded]
```

At the next level of abstraction the situation becomes worse because the DTPA's experience is spread over more of event types. The planning assistant has the same experience with `E1-Load(Here(Car))` and `W1-Load(Here(Stuff))`, so here even the maximum likelihood estimate fails to give a clear choice.

```
[occ so-far 25 E1-Load(Here(Car))]
[occ so-far 10 E1-Load(Here(Car)) Loaded]
[occ so-far 25 E1-Load(Here(Stuff))]
[occ so-far 5 E1-Load(Here(Stuff)) Loaded]
[occ so-far 25 W1-Load(Here(Car))]
[occ so-far 15 W1-Load(Here(Car)) Loaded]
[occ so-far 25 W1-Load(Here(Stuff))]
[occ so-far 10 W1-Load(Here(Stuff)) Loaded]
```

At the third level of abstraction, we are back to the situation described previously, so we can make a choice. The planning assistant chooses to have the manager load the oranges.

```
[occ so-far 100 E1-Load]
[occ so-far 35 E1-Load Loaded]
[occ so-far 100 W1-Load]
[occ so-far 65 W1-Load Loaded]
```

2.4.3 Causing an Event

The DTPA always chooses an action event type that it can cause. Action event types are event types that occur because some agent runs a program. Action events that the planning assistant can execute include the program that makes them occur. Other action event instances represent other agents running programs. For example, the planning assistant itself cannot load oranges, so the planning assistant's representation of such action event instances has no program associated with it.

Once the planning assistant has chosen an appropriate action event type, it generates a hypothetical random instance of that event type and constrains this newly generated event instance so that all its parameters match the plan. For example, it may specify that this random instance should occur at the time the planner specified. The DTPA often adds further constraints to the event type it chose when its choice was made at an abstract level. Even when a concrete event type is chosen, the DTPA's event types rarely specify the time the program should start so it uses the planner's temporal specification.

The DTPA generates a random instance of an action event type, which, because it is an action event instance the DTPA can cause, has a program is associated with it. Because it knows that executing the program associated with the event type causes an instance of that event type, it immediately updates the statistics on that event type. It has observed its own creation of an event instance.

The current DTPA can only execute requests in which it sends a message to an agent. As usual, the event instances generated by requests are collected into event types to gather statistics.

The planning assistant hopes that the right agent will receive the request and act on it appropriately, but it does not know this. If the agent does act on the request appropriately, other event instances occur. The planning assistant hopes that one of the event instances that will occur is one of the events specified in the plan. It also hopes that some of the events that will occur are observable so that it can tell that the goal has been achieved. The planning assistant maximizes its expectation that its goals will be fulfilled by maximizing the probability that the request will result in the goal event types. It monitors observable event instance to see if the goal does occur.

Once the planning assistant has performed all the tasks associated with one event in the plan, it performs the same process on the next event. It continues processing the event specified in the plan until none are left.

2.5 Monitoring the Execution of a Plan

The planning assistant does not make observations automatically. It can monitor events in one of two ways: it can cause an instance of an event type called an anticipation, or it can track an event type. When the planning assistant chooses an action, it reasons about events that are likely to result from this action. If the possible event is observable, it anticipates that event. It then assumes that when an anticipated event is observed, that event is the result of its actions. Alternatively, the planning assistant can track all occurrences of an event type by performing appropriate actions whenever sufficient evidence for the event type occurs.

The planning assistant anticipates by generating a hypothetical event instance and waiting until it sees a set of conditions that would lead it to assume that such an event instance has occurred. Usually, It cannot directly observe the event instances that represent the success of its actions. When it cannot observe success directly, it collects the set of possible

consequences of success and anticipates the subset of these events that are observable. When it makes an observation that matches the observations that would lead it to conclude that an anticipated event instance has occurred, it changes the hypothetical event instance it anticipated into an actual event instance of the same type. It then removes the event instance from the list of anticipations and updates the statistics for the event types of which the event instance is a member. By anticipating, the planning assistant can update the statistics on the event type in which it tries an action and the event type in which the action succeeds.

Once the planning assistant has chosen an appropriate request and has executed the action associated with that request, it collects the set of events that the request `might-cause`. It then chooses from this set the subset that is observable.

Example:

The Load actions mentioned above are actually requests to the appropriate agent to load oranges into a railroad car. Therefore, the DTPA expects three reports from the agent. A subset of the DTPA's knowledge about these reports appears below. The planing assistant needs new event types to represent the reports being generated and being received.

```
Load-Rec
Load-Started
Loaded
Loaded-Rep
Loaded-Rep-Rec
```

```
[might-cause W1-Load Load-Rec]
[might-cause Load-Rec Load-Started]
[might-cause Load-Started Loaded]
[might-cause Loaded Loaded-Rep]
[might-cause Loaded-Rep Loaded-Rep-Rec]
[observable Loaded-Rep-Rec]
[Loaded-Rep-Rec(ei) > Loaded(ei)]
```

Using this knowledge, the planning assistant selects all of the event types that might occur given an event instance of the type it just caused

`W1-Load`. These events are: `Load-Rec`,

`Load-Started`,

`Loaded-Rep`,

`Loaded-Rep-Rec`.

It knows only Loaded-Rep-Rec is observable, so it only anticipates an instance of this event type. The planning assistant knows that if it receives a report stating that loading has been completed then an instance of a Loaded-Rep-Rec occurs. From this it can infer that the loading was successfully completed.

Once the planning assistant has selected a set of observable events that the agent's action might cause, it then chooses the event type that makes such an observation most likely. Each observable event type has an anticipation action event type. The DTPA chooses the anticipation event type that is most likely to cause the observation the same way that it chooses the actions that cause the events specified directly in the plan. That is, the planning assistant chooses an appropriate anticipation and executes the program associated with the chosen anticipation event type.

2.5.1 Observations

The planning assistant monitors the effects of its actions to update its statistics so that it will be able to make better predictions in the future. Because monitoring the execution of the plan is not specified in the plan itself, it must assign utility to monitoring. It assumes that the planner will use its services for many subsequent similar activities and that the best possible answers are desired. Unfortunately, the planning assistant cannot compute this utility without knowledge of the future. Instead, it uses a fixed utility.

The planning assistant also runs a process that monitors its sensory inputs. Each time a sensory input occurs the DTPA checks each of the event instances it is anticipation. If the sensory inputs matches the input expected from an anticipated event instance, it updates the statistics on all the event types that contain this event instance and all event type implied by the occurrence of this event instance. All observable event instances that were anticipated as a result of the planning assistant are then removed from the set of event instances that are anticipated. In this way the planning assistant avoids updating the statistics twice for an event instance that caused more than one observable event instance.

Example:

In the previous example, the planning assistant would have chosen the action event type Anticipate-Load-Rep-Rec. The program associated with all instances of this event type puts a pair consisting of the observation expected–a report from the agent performing the action–and an instance of the event type Load-Rep-Rec. Using the axiom from the previous example, the planning assistant infers that a Loaded-Rep-Rec occurred and therefore that an instance of the Loaded event type occurred. It updates the number of successes for the W1-Load action event type because an instance of that event type lead to the loading.

2.5.2 Tracking Events

The planning assistant tracks event types by noting that a certain set of conditions is sufficient cause to assume that an instance of an event type has occurred. When it sees such a set of conditions, it generates a random instance of the event type it is tracking and updates the statistics on the event type. By tracking event types, the planning assistant can collect information about event types when it does not expect a particular instance of that type.

Tracking could also be used for other purposes. For example, it could reason that a likely observation resulting from requesting a particular inept engineer to traverse a segment of track is that the train will wreck. The planning assistant could anticipate an accident and notify the planner if it observes the accident. To be able to use this kind of information, the planner using the DTPA would have to be able to accept reports that its plan is in trouble.

3 Conclusion

The planning assistant described above avoids the three pitfalls of decision theoretic planners. It avoids the first by relying on the planner to ensure that its choices are independent. It also arranges the probabilities for which it has information into a hierarchy allowing it to find appropriate information quickly.

The planning assistant does not require the person programming it to know the probability of effects given actions. Instead, it infers these probabilities from observed effects of the actions. It gathers these observations by monitoring the plans it executes.

Finally, because it assists a planner, it can assume that the choices it makes are also independent in terms of utility. This allows it to treat each event it tries to achieve as a simple goal. As Haddawy and Hanks [1990] show, such simple goals can be captured by constant utilities.

These solutions are not a panacea. Each introduces new problems into using decision theory for planning.

Because probabilities are inferred from statistics, the DTPA places an even higher premium on knowledge of probability. Only when it has simple choices, few alternatives and much experience, can it choose correctly. It attempts to deal with the lack of information by arranging information in a hierarchy and using it as an abstraction hierarchy. A better use of the statistical information it has would improve its performance but it will still need to focus its attention before applying probability.

Another limitation on the planning assistant's behavior is its reliance on the simple utility structure. The ability to trade off likely success on a low probability goal against likely failure on a high probability goal would greatly improve its performance. The current DTPA cannot make such tradeoffs because it concentrates on one plan event at a time. It focuses its knowledge this way but it cannot switch tasks if one proves too difficult. It cannot shift to another part of the plan because presumably all of the steps in the plan are necessary

for completion. One solution to this problem would be to have the planning assistant give up if it cannot find a reasonable way to cause an event specified in the plan and have the planner generate a new one that does not contain this event.

4 References

[Abadi and Halpern, 1989] M. Abadi and J. Y. Halpern, "Decidability and expressiveness of first-order logics of probability," Computer Science Technical Report RJ 7220 (67987), IBM Research, Almadan Research Center, 1989.

[Allen and Miller, 1991] James Allen and Bradford Miller, "The RHET System: A Sequence of Self-Guided Tutorials," Computer Science 325, University of Rochester, 1991.

[Allen *et al.*, 1991] James F. Allen, Henry A. Kautz, Richard N. Pelavin, and Josh D. Tenenberg, *Reasoning About Plans*, Morgan Kaufman Publishing Co., San Mateo, CA, 1991.

[Allen and Schubert, 1991] James F. Allen and Lenhart K. Schubert, "The TRAINS Project," Computer Science 91-1, University of Rochester, 1991. TRAINS Technical Note.

[Cooper, 1987] Gregory F. Cooper, "Probabilistic Inference Using Belief Networks is NP-Hard," Technical Report KSL-87-27, Stanford Univeristy, Stanford, California 94305, May 1987.

[Fikes and Nilsson, 1971] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, 2:189–205, 1971.

[Haddawy and Hanks, 1990] Peter Haddawy and Steve Hanks, "Issues in Decision-Theoretic Planning: Symbolic Goals and Numeric Utilities," in *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 48–58, 1990.

[Martin, 1993] Nathaniel G. Martin, *Using Statistical Inference to Plan Under Uncertainty*, PhD thesis, University of Rochester, Computer Science Department, Rochester, NY 14627, 1993.

[Pearl, 1988] Judea Pearl, *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, Inc, San Mateo, CA, 1988.

[Wellman, 1990] Michael P. Wellman, "The STRIPS assumption for planning under uncertainty," in *American Association of Artificial Intelligence National Conference 1990*, pages 198–203, 1990.

[Wellman and Doyle, 1992] Michael P. Wellman and Jon Doyle, "Modular Utility Representation for Decision-Theoretic Planning," in *Proceedings of the First International Conference on AI Planning Systems*, pages 236–242, 1992.

*U.S. GOVERNMENT PRINTING OFFICE: 1997-509-127-61008

MISSION OF ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.